Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term:

Display: 10 **Documents in Display Format:** - **Starting with Number** 1

Generate: Hit List [✓ Hit Count] Side by Side Image

Search Clear Interrupt

Search History

DATE: Wednesday, August 15, 2007 **Purge Queries** Printable Copy Create Case

Set Name side by side	**Query**	**Hit Count**	**Set Name** result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L17	L14 and l16	99	L17
L16	L10 and (transaction with data or transaction near data or transaction adj data)	4469	L16
L15	L14 and (key with mask or key near mask or key adj mask)	6	L15
L14	wildcard and key and element with value	627	L14
L13	L12 and (key with mask or key near mask or key adj mask)	1	L13
L12	L10 and wildcard and key and element with value	289	L12
L11	L10 and (wildcard with key with element with value)	0	L11
L10	L9 and (data with elements or data near elements or data adj elements)	20470	L10
L9	L8 and values and parameters	91853	L9
L8	(relational or relation) and (key or column)	404335	L8
L7	707/102	10398	L7
L6	707/1	9949	L6
L5	707.clas.	57568	L5
L4	705.clas.	52753	L4

L3	705/35	3192	L3
L2	705/30	1345	L2
L1	705/1	7479	L1

END OF SEARCH HISTORY

```
  L17: Entry 93 of 99                    File: USPT                   Jan 14, 1997
```

US-PAT-NO: 5594899
DOCUMENT-IDENTIFIER: US 5594899 A

TITLE: Operating system and data base having an access structure formed by a plurality of tables

DATE-ISSUED: January 14, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Knudsen; Helge	Oakville			CA
Chong; Daniel T.	Woodbridge			CA
Yaffe; John	Mississauga			CA
Taugher; James E.	Mississauga			CA
Robertson; Michael	Mississauga			CA
Plazak; Zbigniew	Etobicoke			CA

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Amdahl Corporation	Sunnyvale	CA			02

APPL-NO: 08/347588 [PALM]
DATE FILED: December 1, 1994

PARENT-CASE:
CROSS-REFERENCE TO RELATED APPLICATIONS This application is a continuation of Ser. No. 08/029,902, filed Mar. 11, 1993, now abandoned, which is a divisional of Ser. No. 07/968,237, filed Oct. 29, 1992, now abandoned, which is a continuation of Ser. No. 07/830,548, filed Jan. 31, 1992, now abandoned, which is a continuation of Ser. No. 07/450,298, filed Dec. 13, 1989, now abandoned, which is a continuation-in-part of Ser. No. 07/402,862, filed Sep. 1, 1989, now abandoned under the title "OPERATING SYSTEM AND DATA BASE USING TABLE ACCESS METHOD".

INT-CL-ISSUED: [06] G06F 17/30

INT-CL-CURRENT:

TYPE	IPC	DATE
CIPS	G06 F 17/30	20060101
CIPS	G06 F 9/44	20060101

US-CL-ISSUED: 395/600; 364/DIG.2, 364/974, 364/974.4, 364/972.3, 364/958, 364/958.1

US-CL-CURRENT: 707/2; 707/E17.038

FIELD-OF-CLASSIFICATION-SEARCH: 395/600

See application file for complete search history.

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4893232	January 1990	Shimaoka et al.	364/200
☐	4918593	April 1990	Huber	364/200
☐	5133068	July 1992	Crus et al.	395/600

OTHER PUBLICATIONS

"Table Storage Architecturee for the OS/2 Extended Edition Database Manager", IBM Technical Disclosure Bulletin, vol. 32, No. 5A, Oct. 1989, pp. 30-32.
M. Papazoglou, "An Extensible DBMS for Small and Medium Systems", IEEE Micro, vol. 9, No. 2, Apr. 1989, pp. 52-68.
A. Brown et al., "Data Base Management for HP Precision Architecture Computers", Hewlett-Packard Journal, vol. 37, No. 12, Dec. 1986, pp. 33-48.
D. J. Haderle et al., "IBM Database 2 Overview", IBM Systems Journal, vol. 23, No. 2, 1984, pp. 112-125.

ART-UNIT: 236

PRIMARY-EXAMINER: Kriess; Kevin A.

ASSISTANT-EXAMINER: Chaki; Kakali

ATTY-AGENT-FIRM: Fliesler, Dubb, Meyer & Lovejoy

ABSTRACT:

An object access system for retrieving objects in response to requests identifying requested objects, the system comprising an access structure consisting of a plurality of tables where each table is identified by a unique table identifier and has a plurality of rows where each row has a plurality of fields and is identified by a unique primary key in one of the fields and where a field may also contain objects; a row index for each table, each row index having entries ordered on the primary key of the rows in the table where each entry points to a row of fields in the table; a table index ordered on the table identifier for the tables, the table index having an entry for each table which point to the row index for that table and access means, responsive to requests for an object having an associated table identifier and primary key, for searching the table index for the requested table identifier and for retrieving from the table index entry for the requested table identifier the pointer to the row index for the requested table identifier, searching the pointed to row index for the requested primary key and retrieving from the row index entry for the requested primary key the pointer to the row of fields and searching the pointed to row of field for the requested object and retrieving the requested object.

2 Claims, 40 Drawing figures

Previous Doc Next Doc Go to Doc#

L17: Entry 93 of 99 File: USPT Jan 14, 1997

DOCUMENT-IDENTIFIER: US 5594899 A
TITLE: Operating system and data base having an access structure formed by a plurality of tables

Abstract Text (1):
An object access system for retrieving objects in response to requests identifying requested objects, the system comprising an access structure consisting of a plurality of tables where each table is identified by a unique table identifier and has a plurality of rows where each row has a plurality of fields and is identified by a unique primary key in one of the fields and where a field may also contain objects; a row index for each table, each row index having entries ordered on the primary key of the rows in the table where each entry points to a row of fields in the table; a table index ordered on the table identifier for the tables, the table index having an entry for each table which point to the row index for that table and access means, responsive to requests for an object having an associated table identifier and primary key, for searching the table index for the requested table identifier and for retrieving from the table index entry for the requested table identifier the pointer to the row index for the requested table identifier, searching the pointed to row index for the requested primary key and retrieving from the row index entry for the requested primary key the pointer to the row of fields and searching the pointed to row of field for the requested object and retrieving the requested object.

Brief Summary Text (6):
The development of applications programs by software engineers is a complicated task. This complication arises in part because of environmental parameters, like the variety of data types, hardware types, operating system types, program auditing techniques, and other details. Computer programming languages have developed to handle all of these environmental parameters, usually by requiring explicit recognition of the parameters in the code. Thus, the typical application programmer must contend with data definitions, editing and validation of input data, selection and ordering of data available to the program, looping constraints on the system for the program, output editing and validation conventions, error processing, program auditing, and other complicated tasks in addition to the basic algorithm for accomplishing the application in mind.

Brief Summary Text (7):
These environmental parameters also complicate the running of programs written with high level languages. These programs must be compiled or loaded prior to execution, during which time all the physical resources, data, and rules associated with the application must be bound together.

Brief Summary Text (10):
Accordingly, there is a need for an operating system, data base, and data access method which will allow application programmers to be free of complications caused by environmental parameters.

Brief Summary Text (12):
The present invention provides an operating system, data base, and access method which pushes data definitions, input editing and validation, selection and

ordering, looping, output editing and validation, error processing, and auditing down into the data access method, thereby freeing the application programmer of explicit recognition in his program of these environmental parameters.

Brief Summary Text (14):
The access structure consists of a plurality of tables, each table having a plurality of rows, and each row having a plurality of fields. Each row in the access structure is identified by a unique primary key in one of the fields of the row and by a table identifier. Objects that are retrievable through the access structure are stored as fields in the tables. Tables in the access structure can be further divides into subtables, where each subtable is identified by a table parameter. Tables are identified by a table name and any table parameters that have been assigned to the table.

Brief Summary Text (15):
The access method maintains indexes into the tables stored in the access structure. The indexes are first ordered on the table name, and then ordered on the parameter or parameters associated with a given table. Finally, the indexes are ordered on the primary key of each row within a table.

Detailed Description Text (8):
FIG. 2 shows a conceptual block diagram of the HURON data processing system. The system includes a transaction processor 20, a table access machine 21, and a table data store server 22 which is coupled to direct access storage device 23. The table access machine 21 also drives a screen server 24 which is coupled to a user terminal 25, and an import/export server 26 coupled to other sources of data files 27 and 28.

Detailed Description Text (11):
Through the table access machine 21, physical data stored in direct access storage devices 23, 30, 27, 28 are presented to the transaction processor 20 as if it were stored in the table data store system.

Detailed Description Text (24):
Table 1 shows a sample rule named LEAPYEAR, which has one argument names YEAR. LEAPYEAR is a function because it RETURNs a value--either `Y` or `N`--as a result of execution.

Detailed Description Text (26):
Table 2 shows a more complex rule. The two GET statements verify that the month referred to by the parameter MM occurs in the table MONTHS and that the day referred to by the parameter DD is less than or equal to the maximum for that month. Failure of a GET statement causes the exception GETFAIL, and the exception handler produces a message and returns the value `N`.

Detailed Description Text (27):
Table 3 shows the table MONTHS, which the rule VALID.sub.-- DATE refers to. note the two columns for the numbers of days in a month for leap years and non-leap years.

Detailed Description Text (33):
The rule header gives a name to the rule and defines its parameters (if any). Parameter passing between rules is by value: a called rule cannot alter the value of a parameter. The data representation of a parameter is dynamic: it conforms to the semantic data type and syntax of the value assigned to it. The scope of a parameter is the rule in which the parameter is defined. Table 6 gives an example of a rule header.

Detailed Description Text (35):
Local variables are declared below the rule header. The scope of a local variable

is the rule in which it is defined and any descendant rules. A local variable can be assigned an arithmetic value or a string and can be used anywhere in an action. The data representation of a local variable is dynamic: it conforms to the semantic data type and syntax of the value assigned to it. Local variables are initialized to null (i.e., zero, the logical `N`, or the null string, depending on the usage). Table 7 gives an example of a local variable declaration.

Detailed Description Text (37):
Conditions, which are logical expressions evaluated for their truth value, determine the flow of control in a rule. Conditions are evaluated sequentially, and, if one of them is satisfied, the actions corresponding to it are executable, and no further conditions are evaluated. If there are no conditions (as in the rule in Table 4), then the rule's actions are executable.

Detailed Description Text (38):
Table 8 gives some examples of conditions. REMAINDER and SUPPLIER.sub.-- LOC are functions which return values. Although one of them is a system supplied standard routine and one is a user routine, there is no distinction in the invocation.

Detailed Description Text (39):
Note: the example rules in earlier tables show that the part of a rule that contains conditions also contains a Y/N Quadrant, which displays Y/N ("yes/no") values. The Y/N values coordinate conditions and actions. Huron supplies the Y/N values, however, not the user. The function of the Y/N Quadrant will become clear in the section on actions.

Detailed Description Text (47):
There are two kinds of assignment statement. In simple assignment, a single value is assigned to a field of a table or to a local variable. Table 10 shows simple assignment.

Detailed Description Text (48):
In assignment-by-name, all the values of the fields of the table on the right are assigned to identically named fields of the table on the left. Table 11 shows an example. Assignment-by-name is a convenient way of assigning all the values of fields of a screen table to fields of a data table, or vice versa.

Detailed Description Text (54):
The CALL statement can invoke a rule directly by referring to the rule by its name or indirectly by referring to a field of a table or to a parameter which contains the name of the rule.

Detailed Description Text (55):
The first two examples in Table 14 invoke the rule SELECT.sub.-- RENTAL directly. Note that arguments in CALL statements can be passed by an argument list or by a WHERE clause. A parameter keyword in a WHERE clause must be the name of a parameter in the rule header of the called rule. The calls in the first two examples have identical effects, but the calls use different parameter passing mechanisms. All parameters must be specified when a rule is called.

Detailed Description Text (56):
The last example in Table 14 invokes a rule indirectly. The value of the field ROUTINE in the table PFKEYS is the name of the rule to be executed.

Detailed Description Text (61):
Table parameters can be specified in list form or in a WHERE clause (the two forms correspond to the two methods of parameter passing in the CALL statement).

Detailed Description Text (64):
In the first example of Table 15, the GET statement retrieves the first occurrence

in the table STUDENTS. In the second example, the GET statement retrieves the first occurrence in the table STUDENTS whose field STUDENT# has a value equal to 810883. In the third example, the GET statement retrieves the first occurrence in the table MONTHS whose field MONTH has a value equal to the value of MM and whose field DAYS has a value greater than or equal to the value of DD. In the fourth example, the GET statement retrieves the first occurrence in the table EMPLOYEES whose field

Detailed Description Text (69):
As with all table access statements, parameters can be specified in an argument list or in a WHERE clause. Selection on fields is also specified in the WHERE clause, as in Table 17.

Detailed Description Text (70):
A WHERE clause in a FORALL statement has the same effect as in a GET statement. In the first example of Table 17, the FORALL statement retrieves all occurrences in the table CARS whose field MODEL has a value equal to the value of MDL and whose field YEAR has a value equal to the value of YY.

Detailed Description Text (71):
A WHERE clause can refer to the current table with an asterisk (*). In the second example of Table 17, the FORALL statement retrieves all occurrences in the table EMPLOYEES for which the field HIREDATE has a value greater than the value of the field BIRTHDATE plus 40.

Detailed Description Text (73):
When a FORALL statement is executed, table occurrences are retrieved in primary key order, unless a different order is specified by one or more ORDERED clauses. In the example of Table 18, the occurrences will be presented sorted by descending values of the field PRICE, then by ascending values of the field MODEL, and then by ascending values of the primary key LICENSE (the default for ordering is ASCENDING).

Detailed Description Text (77):
The INSERT statement adds a new occurrence to a table in the database. No field selection is possible: the WHERE clause can only specify parameter values.

Detailed Description Text (78):
Occurrences within a table must have unique primary keys. An attempt to insert an occurrence with a primary key that already exists will cause the INSERTFAIL exception.

Detailed Description Text (79):
Table 19 gives examples of the INSERT statement. Note that, in the second example, CITY is a parameter. The third example shows another way to specify the same parameter.

Detailed Description Text (81):
The REPLACE statement updates an occurrence in the database. No field selection is possible: the WHERE clause can only specify parameter values.

Detailed Description Text (82):
If the occurrence does not exist, the REPLACEFAIL exception is signaled. In order to alto the primary key value of the occurrence, it is necessary to DELETE the old occurrence and INSERT the new one.

Detailed Description Text (83):
Table 20 gives examples of the REPLACE statement. Note that, in the second example, CITY is a parameter. The third example shows another way of specifying the same parameter.

Detailed Description Text (85):
The DELETE statement removes an occurrence from a table in the database. A WHERE clause can specify field selection on the primary key field if the relation specified is equality. No other field selection is allowed. A WHERE clause can specify parameter values, as usual.

Detailed Description Text (86):
If the primary key is specified in a WHERE clause, then that occurrence is deleted. If no primary key is specified, then the occurrence referred to by the primary key in the table template is deleted. If the occurrence does not exist in the table, the DELETEFAIL exception is signaled.

Detailed Description Text (107):
The SCHEDULE statement allows asynchronous processing by allowing a rule to be queued for execution independently of the current transaction. The rule to be executed must exist when the SCHEDULE statement is executed. The name of a queue can be specified by an optional TO clause. Definition of queues is handled by system parameters and is not done within the rule language.

Detailed Description Text (111):
Like the CALL statement, the TRANSFERCALL statement can invoke a rule directly by referring to the rule by its name or indirectly by referring to a field of a table or to a parameter which contains the name of the rule. The first two examples in Table 27 invoke the rule SELECT.sub.-- RENTAL directly, and the last example invokes the rule whose name is the value of the field ROUTINE of the table PFKEYS.

Detailed Description Text (114):
Like the CALL statement, the EXECUTE statement can invoke a rule directly by referring to the rule by its name or indirectly by referring to a field of a table or to a parameter which contains the name of the rule. The first two examples in Table 27 invoke the rule SELECT.sub.-- RENTAL directly, and the last example invokes the rule whose name is the value of the field ROUTINE of the table PFKEYS.

Detailed Description Text (134):
CONVERSION--value contains invalid data for syntax or cannot be converted to target syntax

Detailed Description Text (137):
DELETEFAIL--key for DELETE statement does not exist

Detailed Description Text (142):
INSERTFAIL--key for INSERT statement already exists

Detailed Description Text (144):
OVERFLOW--value is too big to be assigned to target syntax

Detailed Description Text (145):
REPLACEFAIL--key for REPLACE statement does not exist

Detailed Description Text (149):
UNASSIGNED--a field of a table that has not been assigned a value has been referenced

Detailed Description Text (150):
UNDERFLOW--value is too small to be represented in target syntax (mostly exponential errors)

Detailed Description Text (151):
VALIDATEFAIL--validation exit requested through validation exit key

Detailed Description Text (156):
Each element of an expression has a syntax and a semantic data type. The syntax describes how the data is stored, and the semantic data type describes how the element can be used.

Detailed Description Text (158):
The syntax for values of a field of a table is specified in the table definition. The maximum length of the field is also specified in the table definition.

Detailed Description Text (161):
The semantic data type for values of a field of a table is specified in the table definition. The semantic data type determines what operations can be performed on values of the field. Operators in the language are defined only for meaningful semantic data types. For example, negating a string or adding a number to an identifier are invalid operations (consider adding 3 to a license plate number)

Detailed Description Text (171):
The relational operators for equality and inequality (=, =) allow any two operands of the same semantic data type. These operators allow two operands of different semantic data types if the types are identifier and string or identifier and count.

Detailed Description Text (172):
The relational operators for ordering (<, <=, >, >=) allow any two operands of the same semantic data type except logical. Values of type logical are not permitted in comparisons which involve ordering.

Detailed Description Text (175):
The result of a comparison is always a logical value (`Y` or `N`).

Detailed Description Text (178):
A value of any semantic data type can be assigned to a field of the same semantic data type.

Detailed Description Text (179):
A value of type identifier can be assigned to a field of type count (and vice versa).

Detailed Description Text (180):
A value of type identifier can be assigned to a field of type string (and vice versa).

Detailed Description Text (181):
A value of type string can be assigned to a field of type quantity (and vice versa).

Detailed Description Text (182):
A value of type string can be assigned to a field of type count (and vice versa).

Detailed Description Text (197):
The rule COUNT in Table 32 is a generic routine that determines the sum of a field over all occurrences. This rule generalizes the rule COUNT.sub.-- CARS in Table 4 so that it sums any field of any table (more precisely, any table without parameters): it receives the name of the table in the parameter TABLEREF, an it receives the name of the field in the parameter FIELDREF. The parentheses around the names TABLEREF and FIELDREF signify indirection.

Detailed Description Text (199):
The value of TABLEREF will be `PARTS`, the value of FIELDREF will be `PRICE`, and the call will find the sum of the prices of all parts.

Detailed Description Text (224):
The following list of names are reserved by the system as key words in the rule language.

Detailed Description Text (226):
The table data store stores data in relational tables according to the unique table data structure. This structure can best be understood by understanding how tables are built through the table access machine.

Detailed Description Text (230):
Data Stores are in a B+ Tree relational data structure

Detailed Description Text (231):
Since HURON is a transaction system, a user's access to a large amount of data will only affect their dependent region.

Detailed Description Text (240):
The syntax specifications of parameters and fields describe how the data is stored.

Detailed Description Text (245):
TYPE: The table type specifies the access method. Table Data Store (`TDS`) is the default value and is used as the reference template. Each table type has an associated template for display. When the table type is changed, the corresponding template is displayed by pressing any of the function keys or the ENTER key. Valid table types include `TEM` (temporary), `IMP` (import), `EXP` (export), `PRM` (parameter), `SUB` (subview) and `IMS` (IMS) and others defined by a user.

Detailed Description Text (247):
IDGEN: This informs the system that it is responsible for providing unique primary keys for each occurrence.

Detailed Description Text (248):
PARAMETER: The parameter information component is a scrollable area for multiple entries. A maximum of four entries are allowed. This feature allows the system to partition its databases based on a unique field. This mimics a hierarchial structure of data which is more common in the real world than truly relational.

Detailed Description Text (254):
KEY The valid entry is `P` for primary, and blank (non-key field). A table must have one field defined as its primary key. The primary key specification effectively makes the field a required one. Each occurrence in the table is uniquely identified by its primary key value.

Detailed Description Text (255):
RQD The default value for this filed is blank (not required). Other valid entries are `Y` for required or `N` for not required. Inserting or editing an occurrence without proper values in the required fields is not allowed.

Detailed Description Text (256):
DEFAULT The default value of the field, this will be input if the filed is left blank when a new occurrence is being added.

Detailed Description Text (269):
value of "Y" for yes

Detailed Description Text (270):
value of "N" for no

Detailed Description Text (291):
valid lengths range from 1 to 128 bytes for a primary key field and 1 to 256 bytes for other fields.

Detailed Description Text (294):
valid lengths range from 3 to 128 bytes for a primary key field and 2 to 256 bytes for other fields.

Detailed Description Text (311):
PARAMETERS: New parameters can be specified in the subview if there is a corresponding field in the TDS table. Source table parameters can be renamed and the source name specified in the SOURCE PARM.

Detailed Description Text (314):
SRC: This is the source indicator field. Fieldnames can be the same, renamed or unique to the subview table. The source indicator field identifies the status of the field in relation to the source table.

Detailed Description Text (322):
The source of a derived field ought to be a functional rule which returns a value for this field. Applicational

Detailed Description Text (324):
In ADHOC processing the derived fields receive values when the table is accessed. for example, when editing the table:

Detailed Description Text (356):
This table EMP.sub.-- AUDIT will contain information about the user and also the values of the important fields such as salary.

Detailed Description Text (359):
Note that this table will use an automatically generated key.

Detailed Description Text (370):
When the subview provides source rules to be executed, these rules are functions returning one value.

Detailed Description Text (387):
The difference between the two dates--HIREDATE and today's date would be obtained using the function DATE.sub.-- DIFFERENCE. This value is then returned anytime the data is access through this subview.

Detailed Description Text (400):
The leftmost column of the screen is reserved for entering line commands. A line command is entered by placing the first letter of the command in the command space on the line. All line commands are processed when a function key or the ENTER key is pressed.

Detailed Description Text (409):
2. Primary Commands and Function Keys

Detailed Description Text (410):
The primary commands are entered in the area provided on the first line of the screen. Most primary commands have corresponding function keys for user convenience. Following is a list of PF keys and their functions and associated primary commands:

Detailed Description Text (424):
The screen definition provides for default keys and will automatically perform scrolling for the user.

Detailed Description Text (427):
If the user provides a fieldname from a screen table within the SCROLL AMOUNT ENTRY area then HURON will allow scrolling values such as M--maximum, P--page, etc . . . , to be invoked without any further coding.

Detailed Description Text (431):
Following is a list of PF keys and their functions:

Detailed Description Text (441):
Following is a list of PF keys and their functions:

Detailed Description Text (445):
Also, the dictionary includes a table named FIELDS (table name) which is a table which includes the attributes of all data elements in the system and has the structure set out in FIG. 4. This table is parameterized on the table name. Therefore, the table access machine generates a view of the table called FIELDS which is limited to the fields of a given table.

Detailed Description Text (447):
This table identifies the parameter associated with each table, if there are any.

Detailed Description Text (449):
Other dictionary tables include ORDERING (table name) as shown in FIG. 7 which defines a set of ordering operations to be implemented upon accesses to the table, EVENTRULES (table name) as shown in FIG. 8 which specifies rules to be executed upon access events to occurrences in the table, @RULESLIBRARY (library name) as shown in FIG. 9 which stores actual object code for executable rules for the session. The parameter library name is a basic method for dividing up the rules library by a variety of names.

Detailed Description Text (450):
The dictionary also includes dictionary tables required for accesses through the servers other than the table data store. For instance, FIGS. 10, 11, and 12 shown the tables IMSTABFIELDS (table name), IMSSEGFIELDS (db name, seg name), and the IMSACCESS (table name) tables which are used by the IMS server. The IMSTABFIELDS table maps the table access method filed name to an IMS field name. The IMSSEGFIELDS table provides the syntax and mapping parameters for an access based on parameters retrieved from the IMSTABFIELDS table. The IMSACCESS table is used by the server to generate actual access sequences for transmission to the IMS data base.

Detailed Description Text (459):
Rule object code is stored in the @RULESLIBRARY table. This table is parameterized by library name and has the rule name as its primary key.

Detailed Description Text (461):
TABLE 46 shows the detailed layout of the rule object code. The header and code sections contain references to objects in the static data area. These references are two byte binary offsets which are relative to the start of the header. The Parameters, Local Variables, Exception Handler Names, Table.Field Names, Rule names and Constants Sections all belong to the static data area.

Detailed Description Text (463):
The header portion of the object code, which is shown in TABLE 47, contains the name of the rule along with various other values which describe the code, static data and modifiable data areas.

Detailed Description Text (464):
The length stored in the header is the number of bytes to the right of the length

value. Thus the total length of the rule is the value in Length plus 28 (the number of bytes to the left of and including the Length value).

Detailed Description Text (467):
The values in the header which describe the static data area's layout are identified, when appropriate.

Detailed Description Text (468):
Parameters

Detailed Description Text (469):
The parameter section of the object code contains the list of names of the formal parameters declared for this rule.

Detailed Description Text (470):
As shown in Table 48, item in the list is a 16 byte name. "Num parms" contains the number of formal parameters declared for this rule, while the "Parm off" value is an offset to the start of this list.

Detailed Description Text (487):
The virtual stack machine is based on a stack architecture. Most virtual machine instructions manipulate the items on the top of the stack. All stack items are four bytes in length. Many items are pointers to values which are either fields of a table, rule constants or temporary values built during execution. As shown in Table 54, all values are comprised of a six byte descriptor, followed by a four byte pointer to the actual data. The data itself is prefixed by a one or two byte length. The stack may contain other information, such as that necessary to implement rule call and return.

Detailed Description Text (488):
For values that are built during execution, the virtual machine maintains a temporary area. When temporary values are popped from the stack, their storage in the temporary area is freed.

Detailed Description Text (493):
The size of the modifiable area is calculated when a rule is translated from source to object code. This size is stored as the "Mod data" value in the rule header. The first time a rule is called during the course of a transaction, its modifiable data area is allocated and set to contain zeros. As the rule is executed, references to fields of tables, local variables and rules are bound by saving their addresses at the designated offset in this rule's modifiable area. The modifiable data area of a rule is deallocated when the transaction terminates.

Detailed Description Text (494):
The modifiable data area is also used to save the names of indirect rule and indirect table.field references. As described in the Rule opcodes section, an indirect reference is denoted by a non-positive offset as an operand to the @RFIELD, @WFIELD, @CALL, @EXEC, @XCALL or @WARG opcodes. The non-positive offset indicates that the name of the rule or table.field being referenced is on the virtual machine stack. This name is saved in the modifiable data area at the absolute value of the offset along with its thread. (See, Tables 57-58).

Detailed Description Text (505):
Arguments to a rule may be passed by position or by name (using a WHERE clause). The code generated for rule calls when parameters are passed by position is very straightforward. First each argument is evaluated and made a temporary on the stack. This is followed by the @CALL (or @FN for a functional rule) opcode which takes the number of arguments and the name of the rule as it's operands. When the @RETURN opcode in the rule being called is executed, all the arguments passed to the called rule are popped. If the called rule is a function, the return value is

left on the top of the stack.

Detailed Description Text (506):
When arguments are passed by name, the code generated is slightly more complicated. Each argument is evaluated and made a temporary, just as it is when parameters are passed by position. However, the name of the formal parameter which each argument corresponds to, is also pushed, immediately after the argument. This results in parameter name followed by parameter value on the stack for each argument. Immediately preceding the @CALL opcode is the @WARG opcode. @WARG massages the name/value pairs on the stack into the same form as when arguments are passed by position. This allows the @CALL opcode to be executed unaware that the arguments were actually passed by name.

Detailed Description Text (511):
The second argument is the name of the table or screen being accessed. This name is always padded out to be 16 characters in length. The third argument contains the list of parameter values for the table. Each element in this list is a four byte pointer to a value. Elements in the parameter list are inserted by the @TAMP opcode.

Detailed Description Text (512):
The fourth argument, which contains the selection that is to be applied to the table, is the most complicated to describe. Essentially it contains a postfix representation of the WHERE clause. It is comprised of one or more terms which are joined by logical operators. Each term begins with a field reference, is followed by an expression (made up of field references, values and arithmetic operators) and ends with a relational operator. Each field reference is 17 bytes long: a one byte binary code indicating that this is a field reference followed by the name of the field. Each value is a one byte binary code for value, followed by a four byte pointer to the value. Every operator is a single byte binary code, representing the particular operator. As with elements in the parameter list, the @TAMP opcode is used to insert references to values in the selection string.

Detailed Description Text (521):
Indirect table and screen names for table access statements behave somewhat differently than described above. After the value of the table or screen name is pushed onto the stack, a special opcode (@TAMN) is generated. It pops this value and copies it into a temporary value on the top of the stack.

Detailed Description Text (531):
FIG. 17 provides an example of execution by the virtual stack machine for calling a rule named "CHECKTRAIL". In this example, the CHECKTRAIL rule includes two parameters, including `TORONTO` and PERSON. The rule object code is stored in the rule library at the location 120. The instruction pointer will be advanced from the top code @CONST `TORONTO` to the last object @CALL 2, offset, where the offset is indicated by the line 121. The offset identifies the position within the static data area of the rule where the rule name CHECKTRAIL with offset 122 is stored. The offset 122 stores a location in the modifiable area of the rule at which the value for the CHECKTRAIL object code is stored. The value points across line 123 to the location in the rule library 124 at which CHECKTRAIL can be found. The stack 125 is impacted as follows:

Detailed Description Text (532):
First, in response to the @CONST opcode, the value of the variable TORONTO is placed in the working area of the vertical stack machine. In response to the @TEMP opcode, a temporary copy of the TORONTO parameter is stored. The stack is loaded with an offset 126 to the value 127 for the temporary copy 128. Next, the parameter `PERSON` is moved into the working area of the virtual stack machine and a temporary copy is made. The stack is loaded with a pointer 129 to the value 130 of the temporary copy of the constant PERSON 131. Next, the @CALL 2, offset rule is

executed. If the offset in the @CALL opcode points to the name and offset in the static data area of the rule to be called, the value which identifies the location in the rule store of the rule to be called is stored in the modifiable data area at the location 132 identified by the offset 122. The instruction pointer in the virtual stack machine is then moved to the location in the rule store 124 at which CHECKTRAIL is found, and CHECKTRAIL is executed.

Detailed Description Text (533):
The rule libraries 101 shown in FIG. T consist of a plurality of rule libraries. When a @CALL opcode is executed, a search for the appropriate rule is carried out by rule name. The rule can be found as rule object code in one of the libraries, as a built in function or sub-routine (e.g., the ENTER key sub-routine which responds to user input), a customer defined external routine which is written perhaps in a different source language like Cobal or PL1, or it could be a local variable reference, rather than a call to an actual rule. Thus, the rule name search proceeds first through the list of local variables in the rule from which the statement was executed, second, through the local rules library which is set up by the user for a given session, next, through an installation rules library which is a standard set of routines established by a customer for a given execution and programming environment, fourth, through a built in routines library which includes all the standard functions of the operating system, fifth, through a system rules library which stores the Huron system rules which are utilized in the data management and other built in operations, and sixth and finally, through a library of external routines.

Detailed Description Text (537):
FIG. 19 shows the table definition built in the data management area in the virtual stack machine. The table name is first hashed into a table name hash table 160. Then entry in the hash table 160 points to a table list 161 which is searched to find the name of the table "FRUIT" of interest. This entry includes a first pointer 162 to a parameter list 163 and a second pointer 164 to a field list 165 for the table. This field list includes a pointer 166 to the row buffer 167 for the occurrence of the table which has the field of interest.

Detailed Description Text (542):
As soon as a buffer for a particular table has been established, it is henceforth available for all rules in the current transaction. The virtual stack machine keeps track of which fields and which tables have been assigned a value, either through a GET or FORALL command or through an assignment statement.

Detailed Description Text (553):
4. Parameters

Detailed Description Text (560):
G: Get with key

Detailed Description Text (561):
N: Get without key

Detailed Description Text (590):
TAM implements the concept of temporary tables, which exist only during a transaction (in the TAM data areas), but does not materialize in any data server like TDS.

Detailed Description Text (596):
In addition, the buffer pointer for a given table T1 points to a set 252 of buffers for the table T1. This set 252 includes an index 253 for triggers and parameters for table 1. This index points to the actual data occurrence buffers 254, 255, 256, and so on, as required. In addition, the index page 253 points to other T1 information, such as selection strings and the like, in page 257. In addition, the

intent list for table T1 is maintained in pages 258 and 259, as required.

Detailed Description Text (599):
FIG. 22 is a chart showing the table types versus the table opcodes which are executable over those table types. The FIG. is basically self-explanatory, except that the table type SUB, which is a sub-view of either a TDS table type 1 or an IMS table type 2. In addition, the IMP type tables and the EXP type tables have a parameter PARM1 which is equal to the PDS member.

Detailed Description Text (600):
FIG. 23 schematically illustrates the CTABLE formation in the table access machine. The CTABLE is a table of characteristics of a corresponding table stored in a buffer 280. This buffer is generated from raw data 281 which is retrieved from the dictionary tables 282 in the table data store. These dictionary tables are in turn maintained in the balanced storage structures 283 of the table data store just as all other data in the system. Thus, the metadata in a CTABLE is stored in the relational, object-oriented data base upon which the transaction is executing.

Detailed Description Text (603):
FIG. 24 is a representation of a source table (T) and a sub-view table (S). A sub-view is a map of some rectangle within the source table. Thus, as can be seen in FIG. 24, the sub-view includes the first and third columns of rows 2, 3 and 5 of the source table T. This sub-view is implemented by the table access machine with one set of data buffers, but two CTABLES. A sub-view is built for each occurrence at interface time. Derived fields, however, are generated only from the source table. Thus, in response to a request in the rule that needs access to the sub-view table S, the table access machine will build a control table for both the source table T and the sub-view table S.

Detailed Description Text (606):
Parameterized tables are implemented in the access model, but not at the occurrence level. Thus, they are executed in a manner similar to sub-views. These parameterized tables allow concatenated keys, improved concurrency, save storage space in the virtual machines, and simplify archiving. They are executed by building a parameter table in the table access machine out of the dictionary in the table data store. This control table is built in response to the "P" call to the table data store from TAM. The table access machine then transfers the parameter occurrences to the executor for entry in the table lookup indexes.

Detailed Description Text (607):
TAM will retrieve data from the table data store from the source table. It builds a buffer for the source table and then will perform the parameter selections prior to sending occurrences to the executor.

Detailed Description Text (608):
Also, sub-view selection can specify a source parameter name on RHS of the operator.

Detailed Description Text (610):
FIG. 25 shows the code and insertions in an intent list for a given operation. See the intent list stores the intent to store the value k=5 and the value k=4. The FORALL statement executed at the end invokes the sort to establish the insertion. Other operators like replace, delete, and get use the intent list in a similar manner.

Detailed Description Text (621):
The parameter list contains, at the given byte offsets, the following information:

Detailed Description Text (623):
Detailed information about the contents of the field and value descriptors, and

selection string term opcodes are given in Tables 72 and 73, respectively:

Detailed Description Text (624):
The LIKE relational operator has two operands: a field name and a pattern. The result of the relation is true if the value of the field in the row matches the pattern. The pattern may include the wildcards: `*` (any string of zero or more characters) and `?` (any one character).

Detailed Description Text (626):
Variable Length Row format begins with a 2-byte row length designator, which gives the length of the entire row including the designator itself. Appended to the row length designator are one or more fields, the first field being a "key" field. Each such field begins with a 1-byte field length indicator if the field length is less than 128 bytes excluding the length indicator (bit 0 clear), or 2-byte field length indicator if the length is greater than 127 bytes (bit 0 set). The value of the field is represented in the given number of bytes. Blanks are removed from the right end of all fields containing syntax characters. If a requested field is not present in the row, the field is assumed to have a null value.

Detailed Description Text (627):
Expanded Variable Length Row format differs slightly, in that no row length designator is used. Also, fixed numbers of bytes are reserved for each field's value, with the length indicator for each field merely indicating how many of those bytes are actually used. The final difference is that if a requested field is not present in the row, the row is not selected unless the field is being compared with a null string.

Detailed Description Text (632):
Thus, a table access can be summarized as beginning with a table access statement in the object code of the executor. The object code thus includes the code which builds the @TAM arguments, and constants to be used by such arguments. This code can be rebuilt individual form at the source level by the rule editor. The table access statement in object code is then translated into a @TAM opcode format. This includes the opcode, table names, parameters, selection strings, ordering information onto the stack. The values are stored in descriptor format, and based on data management data structures. The accesses are made using field names. Next, the TABLE ACCESS MACHINE INTERFACE is executed, which the opcode, table name, parameters, selection strings, and ordering are set out in machine code format.

Detailed Description Text (636):
Objects in a rule which are not bound include rules or data stored in tables. When a rule calls another rule by a name, the rule will include an offset to a position in the modifiable data area of the rule at which the value of the rule to be called can be stored. The executor then searches for the rule by name through the hash tables and its buffers. If it is not found in the hash tables, then a call to the table access machine is executed to insert the buffer and the proper values in the hash tables. When the buffer for the rule to be called is found by the executor, then the value of the buffer is inserted into the modifiable portion of the rule. Similarly, when an object is called by a table and field name by a rule, that table is found by the virtual stack machine through hash tables or libraries. The value for the occurrence of the table in which the called field is found is then stored in the modifiable portion of the rule at the specified offset.

Detailed Description Text (640):
The table data store is the native access method for the object-oriented operating system according to the present invention. Data in this system is stored in a balanced structure where each table is ordered on the primary key within the table. The basic structure of the access method implemented by the table data store is shown in FIG. 27. The table data store consists of a table index page 500. The table index page includes a list of table names with pointers to a parameter index

page 501 where appropriate. Other entries in the table index page point directly to a primary key index 503. Entries in the parameter index page 501 point to a primary key index 502. The primary key index is a balanced binary tree 507 based on the B+ tree organization ordered on the primary key (where index 502 is the top of the tree). The bottom of the B+ tree points to the actual link to data pages which store occurrences for the table.

Detailed Description Text (641):
The parameter index 501 or the table index page 500 can point also to a secondary key index 508. Secondary key index 508 points to lower sets 509 of indexes on a secondary key based on the B+ key structure. These indexes point to the same linked data pages 510 as the B+ tree on the primary key 507.

Detailed Description Text (642):
The page layout for all page types is shown in FIG. 28. The page includes a 32 byte header 520. The header consists of the page number 521, a previous page pointer 522, a next page pointer 523, a page type indicator 524, and a date value 525. A time stamp 526 is included, as well as a transaction I.D. 527, a check point number 528, the number of rows in the page 529, and the size of each row in the page 530. A data area 531 consists of 4,051 bytes of raw data. At the end of the page, a 13 byte record definition area is reserved for VSAM INFORMATION 532 used by the host system.

Detailed Description Text (644):
where LL is equal to a 2 byte length of the row, l is equal to a 1 or 2 byte length for the value, and the value is the actual data entry or a null value.

Detailed Description Text (649):
Accordingly, searches for occurrences in a table by the primary key are carried out very quickly. For searches on non-ordered fields, an entire table must be retrieved to the server for selection or the table access machine for creating a sub-view or the like.

Detailed Description Text (655):
INDEX--the primary key;

Detailed Description Text (662):
INDEX--primary key;

Detailed Description Text (673):
INDEX.sub.-- PRIMARY KEY;

Detailed Description Text (681):
The SYMTAB table contains information about rules, such as local variable, parameters, and global unbound names directly or indirectly called in the rule. The field of the SYMTAB table include:

Detailed Description Text (682):
IDENT--primary key (identifier name);

Detailed Description Text (683):
KIND--parameter, local or global;

Detailed Description Text (685):
INDEX--unique within the kind (key).

Detailed Description Text (686):
Again, the parameters and locals and globals in the object code format can be generated from the SYMTAB table. The TABLE.sub.-- REFS table contains information about the table.field references. The fields include the following:

Detailed Description Text (687):
INDEX--primary key;

Detailed Description Text (699):
EX.sub.-- NUM--the primary key;

Detailed Description Text (709):
INDEX--primary key;

Detailed Description Text (710):
TYPE--the detranslation type (parameter, constant, table.field reference, . . .);

Detailed Description Text (715):
INDEX--primary key;

Detailed Description Text (719):
The first step is shown in FIG. 31. The value "1" is inserted into the DET.sub.-- STACK with the marker X indicating a boundary element. The DET.sub.-- PRED.sub.-- ST is filled with the precedence 10 of the first data element. The DET.sub.-- WORKAREA is empty.

Detailed Description Text (720):
The next step is shown in FIG. 32. In this step, the value constant 2 is entered as an element of the DET.sub.-- STACK and its precedence is set on the DET.sub.-- PRED.sub.-- ST as a value 10. The work area remains empty. In the next step, the @ADD operator is encountered. In this step, the constant 2 is moved to the work area and its precedence removed from the stack. Next, the "+" operator is added to the work area stack as part of the element including the constant 2. Finally, the constant 1 is moved from the stack to the work area as part of the same element as the constant 2 and the "+". These steps are sequentially shown in FIGS. 33A, 33B, and 33C.

Detailed Description Text (721):
In FIG. 34A, the contents of the work area are transferred back to the main stack. In FIGS. 34B, 34C, 34D and 34E, the TOKENS table is filled. This happens in response to the @SETL opcode which inserts the first and second tokens as shown in FIG. 34B. Thus, the fields of the tokens table for the first token is filled with the value L, an identifier I. It is a one character string with zero positions to the right of the decimal point. It is not a part of the table.field reference. The next entry 2 is the equal sign. It is an operator with one character length with no points to the right of the decimal point and not part of a table.field reference. The contents of the stack are then reordered into the work area to the in-fix form as it appears in the rule language as shown in FIG. 34C. Finally, the balance of the TOKENS table is build from the work area entries as shown in FIG. 34D. The 1 is taken from the top of the work area and inserted as the third entry in the TOKENS table. The "+" operator is inserted as the fourth entry. The constant 2 is inserted as the fifth entry, and the ";", which is the end of the line, is inserted as the last entry. After this occurs, the stack, the precedence table and work area are cleared as illustrated in FIGS. 34E.

Detailed Description Text (724):
The object-oriented operating system of the present invention offers a new generation of application system environment which could substantially enhance programmer efficiency. Some of the key technologies implemented in this invention include the central active on line repository of tables which maintains definitions of data and programs, and links all resources necessary at execution time without regard to where in the storage system the resources reside. It enables dynamic data driven structure in the environment of the system to operate without performance limitation of tradition interpretive systems. Further, the operating system of the

present invention is not limited to any particular hardware or control program environment. Programming developed on one node may be moved easily to any other platform of the system, such as from a main frame to a work station. This makes it possible to support a broad user base with a single programming structure.

Detailed Description Text (728):
The operating system implements data driven programming. This makes it easy to store all program parameters in the data base. Further, clear separation between data and programs can be easily accomplished.

Detailed Description Text (729):
The data representation provided by the operating system provides extensions to the traditional relational model to offer the functionality of the relational conceptual model together with the performance associated with hierarchical internal models.

Detailed Description Text (733):
This permits the extended relational conceptual view of data to be distributed across different physical data base systems, including the table data store, IMS, DB2, and other main frame data base management systems. Also, new programs may be developed using the relational views no matter where the data is located. User does not need to include the calls to specific data base management systems within his programs.

Detailed Description Text (734):
The virtual stack machine with dynamic binding, the on-line dictionary maintained by the table access machine within the native data store, and the extended relational view of data which treats all objects in the system with a common view combine to form an object-oriented operating system that provides a very high level interface for programming and data access, and a complete data processing system for development and production of programs.

Detailed Description Paragraph Equation (11):
EX=====>STE(`table name(parameters)`)

Detailed Description Paragraph Equation (12):
Command line===>EX STE(`tablename(parameters)`)

Detailed Description Paragraph Equation (23):
secondary key value/primary key value/pointer.

Detailed Description Paragraph Table (35):
B (binary) valid lengths are 2 and 4 bytes P (packed decimal) valid lengths range from 1 to 8 bytes, which can hold from 1 to 15 decimal digits the number of decimal digits is specified in the table definition F (floating point) valid lengths are 4, 8, and 16 bytes, for a primary key field, or from 1 to 256 bytes for other fields C (fixed length character string) valid lengths range from 1 to 128 bytes, for a primary key field, or from 1 to 256 bytes for other fields V variable length character string valid lengths range from 3 to 128 bytes, for a primary key field, or from 3 to 256 bytes for other fields storage is reserved for the length specified, but string operations use the current length

Detailed Description Paragraph Table (36):
I (identifier) C (fixed length character string) V (variable length character string) B (binary) P (packed decimal) S (string) C (fixed length character string) V (variable length character string) L (logical) C (fixed length character string of length 1) Possible values: Y (yes) N (no) C (count) C (fixed length character string) V (variable length character string) B (binary) P (packed decimal with no decimal digits) Q (quantity) C (fixed

length character string) V (variable length character string) B (binary) P (packed decimal) F (floating point) ______________________________

Detailed Description Paragraph Table (41):
______________________________ AND ASCENDING CALL COMMIT DELETE DESCENDING DISPLAY END EXECUTE FORALL GET INSERT LOCAL NOT ON OR ORDERED REPLACE RETURN ROLLBACK SCHEDULE SIGNAL TO TRANSFERCALL UNTIL WHERE LIKE Syntax of Rules <rule> ::= <rule declare > <cond list> <action list> <exception list> <rule declare> ::= <rule header> [<local name declaration>] <rule header> ::= <rule name> [<rule header parm list>] ; <rule header parm list> ::= (<rule parameter name> { , <rule parameter name> }) <local name declaration> ::= LOCAL <local name> { , <local name> } ; <cond list> ::= { <condition> ; } <condition> ::= logical value> NOT <logical value> <expression> <relop> <expression> <logical value> ::= <field of a table> <rule parameter name> <function call> <action list> ::= <action> { <action> } <exception list> ::= { <on exception> } <on exception> ::= ON <exception designation> : { <action> } <action> ::= <statement> ; <statement> ::= <assigment> <rule call> <function return> <table access stmt> <sync processing> <display processing> <signal exception> <asynchronous call> <iterative display processing> <assignment> ::= <assignment target> = <expression> <assign by name> <assignment target> ::= <field of a table> <local name> <assign by name> ::= <table ref> .* = <table ref> .* <rule call> ::= CALL <call spec> [<call arguments>] <call spec> ::= <rule name> <rule parameter name> <table name> . <field name> <call arguments> ::= <arg list> WHERE <where arg lists> <where arg lists> ::= <where arg istem> { <and> <where arg item> } <where arg item> ::- <identifier> = <expression> <function return> ::= RETURN (<expression>) <table access stmt> ::= <get stmt> <insert stmt> <replace stmt> <delete stmt> <forall stmt> <get stmt> ::= GET <occ spec> <occ spec> ::= <table spec> [WHERE <where predicate>] <table spec> ::= <table name> [<arg list>] <rule parameter name> [<arg list>] <table name> . <field name> [<arg list>] <where predicate> ::= <where nexpr> { <logical op> <where nexpr> } <where nexpr> ::= { <not> } <where expr> <where expr> ::= <where relation> (<where predicate>) <where relation> ::= <fieldname> <relational op> <where expression> <where expression> ::= [<unary op>] <where expr term> { <add op> <where expr term> } <where expr term> ::= <where expr factor> { <mult op> <where expr factor> } <where expr factor> ::= <where expr primary> [<exp op> <where expr primary>] <where expr primary> ::= (<where expression>) <where field of a table> <rule parameter name> <local name> <function call> <constant> <where field of a table> ::= <where table ref> . <field ref> <where table ref> ::= * <table name> (<rule parameter name>) (<table name> . <field name>) Notice that the <where table ref> production allows a "*" to be specified as the table name. <insert stmt> ::= INSERT <table spec> [WHERE <where arg list>] <replace stmt> ::= REPLACE <table spec> [WHERE <where arg list>] <delete stmt> ::= DELETE <table spec> [WHERE <where arg list>] <forall stmt> ::= FORALL <occ spec> [<table order>] [<until clause>] : <for alist> END <until clause> ::= UNTIL <exceptions> <exceptions> ::= <exception designation> {<or> <exception designation>} <exception designation> ::= <exeption name> [<table name>] <exception name> ::= <identifier> <for alist> ::= { <for action> ; } <for action> ::= <assignment> <rule call> <table access stmt> <display processing> <asynchronous call> <iterative display processing> COMMIT <table order> ::= <table order item> { AND <table order item> } <table order item> ::= ORDERED [<ordering>] <field name> <ordering> ::= ASCENDING DESCENDING <order clause> ::= ORDER <order item> {<and> <order item>} <order item> ::= ORDERED <fieldname> ORDERED ASCENDING <fieldname> ORDERED DESCENDING <fieldname> <sync processing> ::= COMMIT ROLLBACK <display processing> ::= DISPLAY <screen ref> <screen ref> ::= <screen name> <table name> . <field name> <rule parameter name> <screen name> ::= <identifier> <signal exception> ::= SIGNAL <exception name> <asynchronous call> ::= SCHEDULE [<queue spec>] <rule name> [<call arguments>] <queue spec> ::= TO <expression> <iterative display processing> ::= UNTIL <exceptions> <display processing> : {<action>} END <field ref> ::= <field name> (<rule parameter name>) (<table name> . <field name>) <function call> ::= <function name> [<arg lists>] <arg list> ::= (<expression> { , <expression> }) <expression> ::= [<unary op>] <expr term> { <add op> <expr term> } <expr

term> ::= <expr factor> { <mult op> <expr factor> } <expr factor> ::= <expr primary> [<exp op> <exp primary>] <expr primary> ::= (<expression>) <field of a table> <rule parameter name> <local name> <function call> <constant> <field of a table> ::= <table ref> . <field ref> <table ref> ::= <table name> (<rule parameter name>) (<table name> . <field name>) <rule name> ::= <identifier> <function name> ::== <identifier> <rule parameter name> ::= <identifier> <table name> ::= <identifier> <field name> ::= <identifier> <local name> ::= <identifier> <unary op> ::= - + <add op> ::= + - .parallel. <mult op> ::= * / <exp op> ::= ** <logical op> ::= <and> <or> <and> ::= AND & <or> ::= OR .sup. .vertline. <not> ::= NOT <relational op> ::= <rel op> LIKE <rel op> ::= = = > >= < <= <constant> ::= <string literal> <numeric literal>

Detailed Description Paragraph Table (43):
TABLE 33 ____
Define Table - Screen Layout DT EMPLOYEE COMMAND==> TABLE DEFINITION TABLE:EMPLOYEE TYPE:TDS UNIT:educ IDGEN:N PARAMETER NAME TYPE SYNTAX LENGTH DECIMAL EVENT RULE TYPE ACCESS
____ USERID I C 16 FIELD NAME TYPE SYNTAX LENGTH DECIMAL KEY REQ DEFAULT
____ 0 SALARY Q P 3 2 HIREDATE S C 9 0 ADDRESS S V 40 0 CITY S C 20 0 PROV S C 3 0 P.sub.-- CODE S C 7 0 PFKEYS:3=SAVE 12=CANCEL 22=DELETE 13=PRINT 21=EDIT 2=DOC
____
6=RETRIEVE

Detailed Description Paragraph Table (44):
TABLE 34 ______________________________________ DOCUMENTATION SCREEN FOR THE EMPLOYEE TABLE DESCRIPTION OF TABLE:employee UNIT: educ MODIFIED ON: BY: CREATED ON:88.181 BY:educ KEYWORDS: EDUCATION,EMPLOYEE SUMMARY: Table of employees parameterized by USERID DESCRIPTION: ______________________________________ This table contains employee information The education department is responsible for its contents and has designed USERID as a parameter in order to provide each course participant with a copy of the table. PFKEYS: 3=EDIT OBJECT 5=EDIT/VIEW DOCUMENT
____

Detailed Description Paragraph Table (45):
TABLE 35 ____
Screen layout of a subview table DT SUB.sub.-- TABLE COMMAND==> TABLE DEFINITION TABLE:SUB.sub.-- TABLE TYPE:SUB UNIT:educ SOURCE:EMPLOYEE SELECT:USERID = `EDUC` & DEPTNO = 10 PARAMETER NAME TYPE SYN LEN DEC SOURCE PARM ORDER FIELD SEQ
____ FIELD NAME TYP SYN LEN DEC KEY REQ DEFAULT SRC SOURCE NAME
____
MANAGERNO I P 3 0 DEPTNO I B 2 0 PFKEYS: 3=SAVE 12=CANCEL 13=PRINT 15=SAVEON 21=EDIT 22=DELETE 6=RETRIEVE 2=DOC TABLE TYPE CHANGED (PF6 GETS BACK ORIGINAL DEFN). ____

Detailed Description Paragraph Table (46):
TABLE 36 ____
COMMAND==> TABLE DEFINITION TABLE:EMPLOYEE TYPE:TDS UNIT:PER IDGEN:N PARAMETER NAME TYPE SYNTAX LENGTH DECIMAL EVENT RULE TYPE ACCESS
____ USERID I C 100 DEPT.sub.-- CHK V W FIELD NAME TYPE SYNTAX LENGTH DECIMAL KEY REQ DEFAULT
____ 0 SALARY Q P 4 2 HIREDATE S C 9 0 ADDRESS S V 80 0 CITY S C 20 0 PROV S C 3 0 P.sub.-- CODE S C 7 0 PFKEYS:3=SAVE 12=CANCEL 22=DELETE 13=PRINT 21=EDIT 2=DOC
____
6=RETRIEVE

Detailed Description Paragraph Table (48):

TABLE 38 ______
Additional Event Rule for the employee table COMMAND==> TABLE DEFINITION TABLE:EMPLOYEE TYPE:TDS UNIT:PER IDGEN:N PARAMETER NAME TYPE SYNTAX LENGTH DECIMAL EVENT RULE TYPE ACCESS ______ USERID I C 100 DEPT.sub.-- CHK V W EMP.sub.-- AUDIT T W FIELD NAME TYPE SYNTAX LENGTH DECIMAL KEY REQ DEFAULT ______ 0 SALARY Q P 4 2 HIREDATE S C 9 0 ADDRESS S V 80 0 CITY S C 20 0 PROV S C 3 0 P.sub.-- CODE S C 7 0 PFKEYS:3=SAVE 12=CANCEL 22=DELETE 13=PRINT 21=EDIT 2=DOC 6=RETRIEVE ______

Detailed Description Paragraph Table (49):
TABLE 39 ______
COMMAND==> TABLE DEFINITION TABLE:EMPLOYEE TYPE:TDS UNIT:PER IDGEN:Y PARAMETER NAME TYPE SYNTAX LENGTH DECIMAL EVENT RULE TYPE ACCESS ______ FIELD NAME TYPE SYNTAX LENGTH DECIMAL KEY REQ DEFAULT ______ AUDIT.sub.-- NO I B 4 0 P USERID S C 8 0 TRAN.sub.-- DATE S C 8 0 TRAN.sub.-- TIME S C 8 0 EMPNO I P 3 0 LNAME S C 22 2 DEPTNO C B 2 0 SALARY Q P 4 2 PFKEYS:3=SAVE 12=CANCEL 22=DELETE 13=PRINT 21=EDIT 2=DOC 6=RETRIEVE ______

Detailed Description Paragraph Table (51):
TABLE 41 ______
DT TYPE:SUB UNIT:EDUC SOURCE:EMPLOYEE SELECT:DEPTNO = 10 PARAMETER NAME TYPE SYN LEN DEC SOURCE PARM ORDER FIELD SEQ ______ USERID I C 100 0 FIELD NAME TYPE SYN LEN DEC KEY REQ DEFAULT SRC SOURCE NAME ______ 3 0 S MGR# DEPTNO I B 2 0 LGTH.sub.-- EMPLOY I B 4 0 D LGHT.sub.-- EMPLOY 13:PRINT 15:SAVEON 21=EDIT 22=DELETE 6=RETRIEVE 2=DOC ______

Detailed Description Paragraph Table (54):
______ PF Key Command Summary ______ PF1 HELP. PF2 DOC DOCUMENTATION FOR THIS TABLE PF3 SAVE SAVE THE DEFINITION AND LEAVE THE DEFINE TABLE. PF6 RETRIEVE COMMENCES A NEW SESSION BY RETRIEVING THE DEFINITION OF THE NAMED TABLE PF7 SCROLL UP IN THE RULE PF8 SCROLL DOWN IN THE RULE PF9 REDISPLAY PREVIOUS PRIMARY COMMAND PF12 CANCEL LEAVE THE TABLE DEFINE WITHOUT SAVING CHANGES PF13 PRINT PRINT THE DEFINITION PF15 SAVEON SAVE AND CONTINUE PF22 DELETE DELETE THE DEFINITION AND EXIT COPY APPEND THE DEFINITION OF A NAMED TABLE PF21 EDIT SAVE THE DEFINITION & BEGIN AN STE SESSION ______

Detailed Description Paragraph Table (55):
TABLE 44 ______
HURON BUILD SCREEN: employee.sub.-- expense UNIT: acc PF KEYS SCROLL AMOUNT ENTRY DEFAULT CURSOR POSITION ______ UP:7 DOWN:8 TABLE: TABLE: expense.sub.-- data LEFT:10 RIGHT:11 FIELD: FIELD: employee# VALIDATION EXIT:12 HELP:1 REFRESH:24 SCREEN TABLES FOR SCROLL: RULE: TITLE: FOOTING COL ______ acct.sub.-- title 1 1 1 n expense.sub.-- data 5 5 5 y AUTHORIZE 2 PFKEYS: 2=DOCT 6=PAINT 9=DEFHLP 13=PRINT 16=EXCLD 19=DUP 21=DISPLAY ______

Detailed Description Paragraph Table (56):
______ PF Key Function Summary

____ PF1 Help HELP ON SCREEN DEFINITION. PF2 Document DOCUMENTATION FOR THIS SCREEN. PF3 Save SAVE THE DEFINITION AND EXIT. PF6 Paint SAVE AND CALL THE SCREEN PAINTER FOR THE TABLE AT CURSOR POSITION. PF7 SCROLL UP. PF8 SCROLL DOWN. PF9 Define HELP DEFINE HELP SCREEN TO BE ASSOCIATED WITH THIS SCREEN. PF12 Cancel EXIT THE FACILITY WITHOUT SAVING CHANGES. PF13 Print PRINT THE SCREEN. PF16 Exclude EXCLUDE THE TABLE THE CURSOR IS ON. PF19 Duplicate DUPLICATE THE LINE THE CURSOR IS ON. PF21 Display DISPLAY THE SCREEN. PF22 Delete DELETE THE DEFINITION AND EXIT. ____

Detailed Description Paragraph Table (58):
____ PF Key Function Summary
____ PF1 Help HELP ON SCREEN DEFINITION. PF2 Document DOCUMENT FOR THE SCREEN TABLE. PF3 Save SAVE THE DEFINITION AND EXIT. PF4 Add line INSERT A LINE AFTER THE CURSOR. PF5 Cut field CUT & HOLD THE FIELD AT CURSOR. PF6 Add field ADD A FIELD AT THE CURSOR PF7 SCROLL UP. PF8 SCROLL DOWN. PF12 Cancel EXIT THE FACILITY WITHOUT SAVING CHANGES. PF13 Print PRINT THE SCREEN. PF16 Delete line DELETE THE LINE THE CURSOR IS ON. PF17 Paste field RELEASE A CUT FIELD & POSITION AT THE CURSOR POSITION. PF18 Del field DELETE THE FIELD THE CURSOR IS ON. PF19 Copy COPY THE NAMED TABLE DEFINITION. PF22 Delete DELETE THE DEFINITION AND EXIT. ____

Detailed Description Paragraph Table (59):
TABLE 46 ____ Layout of rule object code Header (52 bytes) Parameters (optional) Local Variables (optional) Code for Conditions Code for Actions Code for Exceptions (optional) Exception Handler Names (optional) Rule Names (optional) Table.Field Names (optional) Constants (optional)

____

Detailed Description Paragraph Table (60):
TABLE 47 ____ Layout of rule object code header Value Offset Syntax Purpose ____ Rulename 00 Char (16) Name of the rule Date 16 Char(6) Date of translation Time 22 Char(4) Time of translation Length 26 Bin(2) Length of rest of rule Num parms 28 Bin(2) Number of formal parameters Parm off 30 Bin(2) Offset to local list Num locs 32 Bin(2) Number of local variables Loc off 34 Bin(2) Offset to local list Cond off 36 Bin(2) Offset to conditions Act off 38 Bin(2) Offset to actions Excp off 40 Bin(2) Offset to exception names Function 42 Char(1) Rule is a function (Y/N) TabFld off 43 Bin(2) Offset to t.f names Const off 45 Bin(2) Offset to constants Rule off 47 Bin(2) Offset to rule names Version 49 Bin(1) Object code version# Mod data 50 Bin(2) Size of modifiable data area ____

Detailed Description Paragraph Table (61):
TABLE 48 ____ The internal representation of a parameter ##STR11## ____

Detailed Description Paragraph Table (67):
TABLE 54 ____ The virtual machine representation of a value ##STR17## ____

Detailed Description Paragraph Table (68):
____ Name Opcode #Ops Semantics
____ @ADD 4 0 Add two values together. Pop item1 and item2. Create and push a temporary value of item2 + item1. @SUB 8 0 Subtract one value from another. Pop item1 and item2. Create and push a temporary value of item2 - item1. @MULT 12 0 Multiply two values together. Pop item1 and item2. Create and push a temporary value of item2 * item1. @DIV 16 0 Divide one value by another. Pop item1 and item2. Create and push a temporary value of item2 / item1. @EXP 20 0 Raise one value to the power of another. Pop item1 and item2. Create and push a temporary value of item2 ** item1. @UNM 24 0 Arithmetically negate a value Pop item1. Create and push a temporary value of -(item1). @CAT 28 0 Concatenate two

values together. Pop item1 and item2. Create and push a temporary value of item2 concatenated to item1. @EQ 32 0 Compare two values for equality. Pop item1 and item2. Create and push a temporary value of item 2 = item 1 (`Y" or `N`). @NE 36 0 Compare two values for inequality. Pop item1 and item2. Create and push a temporary value of item2 -= item1 (`Y` or `N`). @LT 40 0 Compare to determine if one value is less than another. Pop item1 and item2. Create and push a temporary value of item2 < item1 (`Y` or `N`). @LE 44 0 Compare to determine if one value is less or equal to another. Pop item1 and item2. Create and push a temporary value of item <= 1 (`Y` or `N`). @GT 48 0 Compare to determine if one value is greater than another. Pop item1 and item2. Create and push a temporary value of item2 > item1 (`Y` or `N`). @GE 52 0 Compare to determine if one value is greater or equal to another. Pop item1 and item2. Create and push a temporary value of item2 >= item1 (`Y` or `N`). @CALL 56 2 Call a procedural rule or builtin. Operand1 is the number of arguments to pass; they have already been pushed onto the stack. Operand2 is an offset to the name of the rule to call; a positive offset is a reference to a name in the static data area. A non-positive offset signifies an indirect rule name, in which case item1 contains the name of the rule to call. On an indirect call, item1 is popped. The caller's environment is saved and execution begins at the start of the called rule. @FN 60 2 Call a functional rule or builtin. Operand1 is the number of arguments to pass; they have already been pushed onto the stack. Operand2 is an offset to the name of the rule to call. The caller's environment is saved and execution begins at the start of the called rule. 64 UNUSED. @RETURN 68 0 Return from a functional or procedural rule. Pop all arguments passed to the called rule and restore the caller's environment. If the called rule was a function, push the return value onto the stack. @UNTIL 72 2 Initialize for the execution of an UNTIL loop. Operand1 is an offset to the opcode to be executed once the loop terminates. Operand2 indicates whether this is an UNTIL . . . DISPLAY loop (1) or just UNTIL loop (0). @PARM 76 1 Push a formal rule parameter onto the stack. Operand1 is the number of the formal parameter. (For a rule with N parameters, the Ith one is numbered (N - I)). Push the parameter value onto the stack. @CONST 80 1 Push a constant onto the stack. Operand1 is an offset to the value in the static data area. Copy the values' descriptor into the temporary area and append a four byte pointer to it. Set the pointer to reference the value's data in the static data area. Push the constant value onto the stack. @TEMP 84 0 Create a temporary copy of a value. If item1 is not a temporary value, create a temporary copy, pop item1 and then push the temporary value. @RFIELD 88 1 Push a field of a table onto the stack. Operand1 is an offset to the table field name; a positive offset references a name in the static data area. A non-positive offset signifies an indirect table.field, in which case item1 is the name of the table and item2 is the name of the field. On an indirect reference, item1 and item2 are popped. Push the field value onto the stack. As the field is being pushed, a check ensures that the field has a value; if not the exception UNASSIGNED is raised. @WFIELD 92 1 Push a field of a table onto a stack. Operand1 is an offset to the table.field name; a positive offset references a name in the static data area. A non-positive offset signifies an indirect table.field, in which case item1 is the name of the table and item2 is the name of the field On an indirect reference, item1 and item2 are popped. Push the field value onto the stack. @AEND 96 0 Mark the end of the current code section. Used to separate the conditions from the and the actions from the exceptions. @AEND is NEVER executed by the virtual machine. @SET 100 0 Assign a field of a table a value. Item1 is a field of a table. Item2 is a value. Set data of item1 to data of item2 and pop both items. @ABN 104 0 Assign commonly named fields in tables. Item1 is the name of the source table. Item2 is the name of the target table. Set the data of all commonly named fields in the target table to data of fields in the

Detailed Description Paragraph Table (69):
source table. Pop item1 and item2. @SCHED 108 2 Schedule the asynchronous execution of a rule. Operand1 is the number of arguments to pass; they have already been pushed onto the stack. Operand2 is an offset to the name of the rule to schedule. ???? what is done with these args and queue name ??? Pop all arguments from the stack. Pop item1, which is the name of the queue for the asynchronous

event and continue. @DROP 112 1 Branch conditionally to an offset within a rule. Pop item1. If item1 is not `Y`, branch to the offset given by operand1. @NEXT 116 1 Branch unconditionally to an offset within a rule. Branch to the offset given by operand1. @BRC 120 1 Branch conditionally to an offset within a rule. Check the return code set as a result of the last @TAM opcode. If it is non-zero, branch to the offset given by operand1. @FORI 124 0 Initialize for the execution of a FORALL loop. @TAM 128 1 Call TAM to perform a table access request. Operand1 contains the number of arguments on the stack. Build a parameter list from these arguments, and call TAM. If the access was other than a FORALL or UNTIL . . DISPLAY, pop the arguments from the stack. If the access was a FORALL or UNTIL . . DISPLAY and the return code is zero (end of occs.), pop the arguments, otherwise leave the arguments on the stack. Save the return code from the TAM call. @TAMP 132 1 Insert a reference into a TAM selection string or parameter list. Item1 is a value which is to be inserted into the selection string or parameter list. Item2 is the selection string or parameter list. Operand1 is an offset in item2 where the reference to item1 is to be inserted. If necessary, create temporary copies of both item1 and item2. Insert the reference to item1 in item2 and pop item1. (item2 will be popped during the course of executing the @TAM opcode). @TAMN 136 1 Copy a table name into a TAM table name argument. Item1 is a table name which is to be passed to TAM. Item2 is a 16 byte blank character string. Operand1 is an offset in item2 where the table name is to be copied. If necessary, create temporary copies of item1 and item2. Copy item1 into item2 and pop item1. (item2 will be popped during the course of execution the @TAM opcode). @SIGNAL 140 0 Raise an exception. Item1 is the name of the exception which is to be raised. Pop item1 and raise the exception. 144 UNUSED. @ACALL 148 1 Call an action from the conditions. Push the offset to the next instruction (the "return address") and branch to the offset given by operand1. @ARETURN 152 0 Return from an action (to the conditions). Item1 is the offset of the next opcode to be executed. Pop item1 and branch to that offset. @EXEC 156 2 Execute a procedural rule. Operand1 is the number of arguments to pass; they have already been pushed onto the stack. Operand2 is an offset to the name of the rule to execute; a positive offset is a reference to a name in the static data area. A non-positive offset signifies an indirect rule name, in which case item1 contains the name of the rule to execute. On an indirect execution, item1 is popped. The enviornment of the current transaction is saved and a new transaction begins at the start of rule being executed. 160 UNUSED. @SETL 164 1 Assign a local variable a value. Item1 is the value to be assigned. Operand1 is the local variable. Set data of operand1 to data of item1 and pop item1. @WARG 168 2 Convert a list of arguments which are passed by name to a list of arguments passed by position. The stack consists of parameter name/parameter value pairs. Operand1 is the number of name/value pairs. Operand2 is an offset to the name of the rule being passed there args; a positive offset is a reference to a name in the static data area. A non-positive offset signifies an indirect rule name, in which case item1 contains the name of the rule. On an indirect reference, item1 is popped. Pop all name/value pairs and push and values in positional order. @NOT 172 0 Logically negate a value. Pop item1. Create and push a temporary value of (item1). 176 UNUSED. 180 UNUSED. @XCALL 184 2 Transfercall a procedural rule. Operand1 is the number of arguments to pass; they have already been pushed onto the stack. Operand2 is an offset to the name of the rule to transfercall; a positive offset is a reference to a name in the static data area. A non-positive offset signifies an indirect rule name, in which case item1 contains the name of the rule to transfercall. On an indirect transfercall, item1 is popped. The environment of the current transaction is discarded, and a new transaction begins at the start of the transfercalled rule.

Detailed Description Paragraph Table (74):
TABLE 59 Source and object code for conditions and actions Rule source code: MAX (X, Y); X < Y; Y N RETURN(Y); 1 RETURN(X); 1 Rule object code: Offset Opcode Operands Comment CONDITIONS 54 : @PARM 1 Push value of X 57 : @PARM 0 Push value of Y 5A :

@LT Pop X,Y and push X < Y 5B : @DROP 62 IF X < Y THEN 5E : @ACALL 67 "Call" action at offset 67 61 : @RETURN ELSE 62 : @ACALL 6C "Call" action at offset 6C 65 : @RETURN ENDIF 66 : @AEND End of conditions ACTIONS 67 : @PARM 0 Push value of Y 6A : @RETURN Return from the rule 6B : @ARETURN End of an action 6C : @PARM 1 Push value of X 6F : @RETURN Return from the rule 70 : @ARETURN End of an action 71 : @AEND End of actions ____________________

Detailed Description Paragraph Table (75):
TABLE 60 ____________________ Source and object code for assignments Rule source code: ASSIGN; LOCAL L;
____________________ L =(G + 1) * 4; 1 T.F = L; 2
____________________ Rule object code: Offset Opcode Operands Comment CONDITIONS 46 : @ACALL 4E Call action 1 49 : @ACALL 5F Call action 2 4C : @RETURN Return from the rule 4D : @AEND ACTIONS 4E : @FN 0 G Push value of G 53 : @CONST 1 Push 56 : @ADD Compute G + 1 57 : @CONST 4 Push 4 5A : @MULT Compute (G + 1) * 4 5B : @SETL L Assign value to L 5E : @ARETURN Return from action 1 5F : @FN 0 L Push value of L 64 : @WFIELD T.F Push value of T.F 67 : @SET Assign L to T.F 68 : @ARETURN Return from action 2 69 : @AEND ____________________

Detailed Description Paragraph Table (77):
TABLE 62 ____________________
Code generated for table accesses. Rule source code: TABLEIO(T);
____________________ GET TABLES; 1 GET TABLES WHERE NAME = T; 2 FORALL @RULESDOCUMENT(LIBID) ORDERED ASCENDING CREATED: 3 END;
____________________ Rule object code: Offset Opcode Operands Comment CONDITIONS 44 : @ACALL 4F Call action 1 47 : @ACALL 59 Call action 2 4A : @ACALL 6F Call action 3 4D : @RETURN Return from rule 4E : @AEND ACTIONS 4F : @CONST "G" Push TAM opcode 52 : @CONST "TABLES" Push table name 55 : @TAM 2 Call TAM (2 args) 58 : @ARETURN Return from act1 59 : @CONST "G" Push TAM opcode 5C : @CONST "TABLES" Push table name 5F : @CONST " " Push dummy parms 62 : @CONST ".NAME . ." Push select str. 65 : @PARM 0 Push value of T 68 : @TAMP 14 Insert reference 6B : @TAM 4 Call TAM (4 args) 6E : @ARETURN Return from act2 6F : @FORI Init for FORALL 70 : @CONST "A" Push TAM opcode 73 : @CONST "@RULESDOCUMENT " Push table name 76 : @CONST " " Push table parm 79 : @FN 0 LIBID Call LIBID 7E : @TAMP 2 Insert reference 81 : @CONST " " Push dummy selstr 84 : @CONST "CREATED A" Push order str. 87 : @TAM 5 Call TAM (5 args) 8A : @BRC 90 Cond. branch 8D : @NEXT 87 Uncond. branch 90 : @ARETURN Return from act3 91 : @AEND
____

Detailed Description Paragraph Table (78):
TABLE 63 ____________________ Code generated for indirect references Rule source code: UNDREFS(R,T); ____________________
CALL R; 1 (T).F = 7; 2 DELETE T; 3 ____________________ Rule object code: Offset Opcode Operands Comment CONDITIONS 44 : @ACALL 4F Call action 1 47 : @ACALL 58 Call action 2 4A : @ACALL 66 Call action 3 4D : @RETURN Return from the rule 4E : @AEND ACTIONS 4F : @PARM 1 Push value of R 52 : @CALL 0 -4 Indirect call 57 : @ARETURN Return from action 1 58 : @CONST 7 Push 7 5B : @CONST "F" Push "F" (field name) 5E : @PARM 0 Push value of T (table) 61 : @WFIELD -18 Push val of indirect t.f 64 : @SET Assign 7 to field 65 : @ARETURN Return from action 2 66 : @CONST "D" Push TAM opcode 69 : @CONST " " Push dummy table name 6C : @PARM 0 Push value of T 6F : @TAMN 0 Put tab name into dummy 72 : @TAM 2 Call TAM (2 args) 75 : @ARETURN Return from action 3 76 : @AEND ____________________

Detailed Description Paragraph Table (79):
TABLE 64 ____________________ (Valid TAM --> Server Requests)
Code Name ____________________ C Ctable request R Release locks G Get N Next P Next Parameter ____________________

Detailed Description Paragraph Table (82):

TABLE 67 (G) ______________________________________ TAM --> Server G request: Byte Offset Length Contents ______________________________________ 0 2 Message length 2 4 Transaction ID 6 1 Request code 7 1 Lock 8 variable Tablename, Parameters, and Key Value data strings Server --> TAM G return: Byte Offset Length Contents ______________________________________ 0 2 Message length 2 2 Return code 4 4 Transaction ID 8 variable Table Row data string

____

Detailed Description Paragraph Table (83):
TABLE 68 (N) ______________________________________ TAM --> Server N request: Byte Offset Length Contents ______________________________________ 0 2 Message length 2 4 Transaction ID 6 1 Request code 7 1 Lock 8 variable Tablename, Parameters, Last Row, Selection, Keyname Ordering, and Keyname .linevert split.op.linevert split. Value data strings Server --> TAM N return: Byte Offset Length Contents ______________________________________ 0 2 Message length 2 2 Return code 4 4 Transaction ID 8 variable Table Rows data string

____

Detailed Description Paragraph Table (84):
TABLE 69 (P) ______________________________________ TAM --> Server P request: Byte Offset Length Contents ______________________________________ 0 2 Message length 2 4 Transaction ID 6 1 Request code 7 1 Lock 8 variable Tablename, Last Parameter Set, and Selection data strings Server --> TAM P return: Byte Offset Length Contents ______________________________________ 0 2 Message length 2 2 Return code 4 4 Transaction ID 8 variable Parameter Sets data string

____

Detailed Description Paragraph Table (86):
______________________________________ Register Contents ______________________________________ 15 address of entry point into Selection Evaluator 13 address of standard save area 01 address of parameter list

____

Detailed Description Paragraph Table (88):
TABLE 71 ______________________________________ (Selection Terms) Term Type Function ______________________________________ Operator 1 - byte arithmetic, relational, or logical opcode Field reference 1-byte opcode (`44`) followed by a 6-byte field descriptor and a 2-byte field number Offset reference 1-byte opcode (`4C`) followed by a 6-byte field descriptor and a 2-byte offset of the field in the row. Used for Temporary, Import, Screen, and IMS tables. Value 1-byte opcode (`48`) followed by a 6-byte value descriptor and a value. Values begin with a 1-byte (length < 128, bit 0 clear) or 2-byte (length > 127, bit 0 set) length prefix.

____

Detailed Description Paragraph Table (89):
TABLE 72 ______________________________________ (6-byte Field and Value Descriptors) Byte Offset Length Contents ______________________________________ 0 1 semantic type 1 1 data syntax 2 2 maximum data length 4 2 number of decimal places

____

Detailed Description Paragraph Table (90):
TABLE 73 ______________________________________ (Hexadecimal Opcodes) Opcode Function ______________________________________ (Arithmetic) `04` + (addition) `08` - (subtraction) `0C` * (multiplication) `10` / (division) `14` ** (exponential) `18` - (unary minus) `1C` .linevert split. .linevert split. (concatenation) (Relational) `20` = (equal) `24` = (not equal) `28` < (less than) `2C` <= (less than or equal) `30` > (greater than) `34` >= (greater than or equal) (Logical) `38` & (and) `3C` .linevert split. (or) `40` (not) (Other) `44` R (reference to i'th field in row) `48` V (value) `4C` O (reference to field at offset in row) `54` @ (parameter replaced in selection and always evaluates to true) `BC` LIKE (relation)

CLAIMS:

1. A computer implemented object orientated relational database management system comprising:

a) a memory means for storing;

1) a plurality of data records;

2) a plurality of tables, each said table having a table name for identifying each said table;

3) a table index having an entry for each said table in said management system where each said entry for a table name including pointers to a location in said memory means of a primary key index and of a secondary key index and where said entries are ordered within said table index upon said table names;

4) at least one primary key index, each primary key index having an entry for at least one primary key where each said entry has a pointer to a location in said storage means of a data record identified by said table name and said primary key and where said entries are ordered within said row index upon said primary keys included in said primary key index;

5) at least one secondary key index having an entry for at least one combination of a primary key and a secondary key where each said entry has a pointer to a location in said storage means of a data record identified by said table name, said primary key and said secondary key where said entries are ordered within said secondary key index first upon said secondary keys and then upon said primary keys for each stationary key in said secondary key index;

b) access means for retrieving a data record identified by a table name, a primary key, and a secondary key, said access means comprising:

1) table search means for retrieving from said storage means said table index, for searching said entries in said retrieved table index for said requested table name and upon finding said entry retrieving said primary key index when no secondary key is identified in said data record and said secondary key index when a secondary key is identified in said record; and

2) first search means for searching said entries in said retrieved primary key index, when said primary index has been retrieved, for an entry for said primary key of said data record and upon finding said entry retrieving said data record from the location in said memory means pointed to in said found entry for said primary key; and

3) second search means for searching said entries in said retrieved secondary key index, when a secondary key index has been retrieved, for an entry having said primary key and said secondary key of said data recorded and upon finding said entry retrieving said data record from the location in said memory means pointed to in said found entry having said primary key and said secondary key.

2. The system of claim 1 wherein:

a) said memory units further comprises:

within a table name entry in said table index a pointer to a parameter index in said storage means; and at least one parameter index having an entry for at least one parameter where each said entry for a parameter has a pointer to a location in said storage means to a primary key index and a secondary key index and where said entries are ordered within said parameter index upon a parameters name for each parameter included in said parameter index; and

b) said access means retrieves a data record identified by a table name, a parameter name, a primary key and a secondary key and wherein:

1) said table search means upon finding said entry retrieves said parameter index from the location in said memory means pointed to in said entry; and

2) parameter search means for retrieving from said storage means said parameter index, for searching said entries in said retrieved parameter index for a parameter name included in a data record to be retrieved and upon finding said entry retrieving said primary key index when no secondary key is identified in said entry and said secondary key index when a secondary key is identified in said entry from the locations of said primary key index and said secondary key index in said memory means pointed to in said parameter name entry.

Previous Doc Next Doc Go to Doc#

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#

☐ Generate Collection Print

L17: Entry 85 of 99 File: USPT Nov 19, 2002

US-PAT-NO: 6484149
DOCUMENT-IDENTIFIER: US 6484149 B1
**** See image for Certificate of Correction ****

TITLE: Systems and methods for viewing product information, and methods for generating web pages

DATE-ISSUED: November 19, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Jammes; Pierre J.	Bellevue	WA		
Franklin; D. Chase	Seattle	WA		
Remington; Darren B.	Issaquah	WA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Microsoft Corporation	Redmond	WA			02

APPL-NO: 08/948453 [PALM]
DATE FILED: October 10, 1997

INT-CL-ISSUED: [07] G06F 17/60

INT-CL-CURRENT:

TYPE	IPC	DATE
CIPP	G06 Q 30/00	20060101

US-CL-ISSUED: 705/26
US-CL-CURRENT: 705/26

FIELD-OF-CLASSIFICATION-SEARCH: 705/26, 705/27, 705/28
See application file for complete search history.

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5491795	February 1996	Beaudet et al.	345/346

☐	5715314	February 1998	Payne et al.	380/24
☐	5745681	April 1998	Levine et al.	705/26
☐	5790116	April 1998	Malone et al.	345/335
☐	5757917	May 1998	Rose et al.	380/25
☐	5848399	December 1998	Burke	705/27
☐	5855015	December 1998	Shoham	707/5
☐	5897622	April 1999	Blinn et al.	705/26
☐	5956487	September 1999	Venkatraman et al.	340/825.06
☐	5970471	October 1999	Hill	705/26
☐	6014638	January 2000	Burge et al.	705/27

ART-UNIT: 2132

PRIMARY-EXAMINER: Smithers; Matthew

ATTY-AGENT-FIRM: Lee & Hayes, PLLC

ABSTRACT:

A system and method for designing and operating an electronic store (1) permit a merchant to organize and advertise descriptions of product inventory over the Internet, (2) permit Web page information to be extracted on-demand from a product inventory database, and (3) permit Web pages to be automatically customized to fit shopping behaviors of individual consumers. A graphical store design user interface of a Web browser displays a hierarchical representation of products and, product groups of an electronic store. A user manipulates icons of the Web browser store design user interface to cause a Web server to modify relationships between products and product groups stored in a product information database. A store designer creates HTML template files, embeds database and customize references within the template files, and assigns template files to groups or products of the electronic store.

The Web server receives requests to access Web pages from consumers using standard Web browsers. The Web server opens a template file related to the requested Web page, creates hyperlinks and other information content by executing database references embedded within the template file, and merges the hyperlinks and content with the template file to generate an HTML Web page to send to the Web browser. The Web server automatically creates additional hyperlinks to any Web pages or products preferred by the consumer by executing customize instructions associated with customize references in a template file. To determine whether any products or Web pages are preferred by a consumer, the Web server examines a traffic analysis database and extracts the consumer's history of accesses to Web pages and orders of products. For each Web page the consumer has accessed, the Web server uses preferred page rules to determine whether the page was accessed with sufficient frequency to generate a hyperlink to the page. For each product the consumer ordered, the Web server uses preferred product rules to determine whether the product was ordered with sufficient frequency to generate a hyperlink to a page offering the product.

17 Claims, 33 Drawing figures

Previous Doc Next Doc Go to Doc#

L17: Entry 85 of 99 File: USPT Nov 19, 2002

DOCUMENT-IDENTIFIER: US 6484149 B1
**** See image for Certificate of Correction ****
TITLE: Systems and methods for viewing product information, and methods for generating web pages

Brief Summary Text (7):
According to one existing method of designing and managing an electronic store, the electronic store is generated by manually assembling and compiling a collection of fixed Web pages. This method generally requires the store designer to have an intimate knowledge of HTML (HyperText Markup Language) to update the content or format of any page. As required by this method, a store designer must learn numerous HTML tags as well as specific parameters for each tag. The store designer uses a standard text editor to edit Web pages by embedding tags, parameters and informational content in text files representing the Web pages.

Detailed Description Text (9):
The enhanced Web browser 112 initiates data transactions with the product information database 116. The enhanced Web browser 112 issues database transaction commands to the Web server 106, which in turn issues those transaction commands to a relational database server 114. In a preferred embodiment, the relational database server 114 utilizes open database connectivity (ODBC).

Detailed Description Text (10):
Relational database servers 114 utilizing ODBC are known in the art. One function of such relational database servers is to provide to application programs a common query interface to interact with multiple database systems having different query interfaces. Methods for providing such common query interfaces are not within the scope of this invention and will not be further discussed.

Detailed Description Text (11):
The Web server 106 passes generic database transaction commands (or queries) received from the enhanced Web browser 112 to the relational database server 114. The relational database server 114 formats the generic database transaction commands 118 received from the Web server 106 as necessary to generate specific database transaction commands required to retrieve, store, or modify information stored in the product information database 116.

Detailed Description Text (12):
The Web server 106 receives requests generated by a standard Web browser 102 on a consumer computer. The standard Web browser 102 provides general capability to request data pages over the World Wide Web by including a URL value in an HTTP-coded request and transmitting that request. Known Web browsers such as Netscape Navigator.TM. 2.2 or Microsoft Explorer.TM. 3.0 are examples of standard Web browsers.

Detailed Description Text (13):
In response to a request for a page, an HTML page engine 126 of the Web server 106 assembles an HTML page. Pages requested by a consumer running a Web browser 102 do not, in many cases, exist prior to the request. The HTML page engine 126 processes the information stored in the HTML template file 108, extracts SQL queries from the

template, and issues these generic product or category queries 122 to the relational database server 114, which in turn issues a specific product or group (i.e., category of products) query 124 to the product information database 116. The HTML page engine 126 receives the results of a product or group query and merges data from the query with a template file 108 to generate an HTML page.

Detailed Description Text (16):
The Web server 106, having access to HTML template files 108 and also having access to data from the product information database 116 via the relational database server 114, provides functionality for operating an electronic store. Neither the enhanced Web browser 112 nor the HTML authoring system 110 are needed to operate an electronic store.

Detailed Description Text (20):
The product information database 116 can be hosted by a number of different relational database systems. For example, existing database products such as Oracle.TM. or Microsoft SQL Server.TM. could each be used to store and manage product information. Even though each such database product may accept a different set of commands for performing similar transactions, the relational database server 114 allows the Web server 106 to communicate with any of these database products using a uniform command interface.

Detailed Description Text (25):
In a preferred embodiment, computer instructions included in the store management control 306 cause relationship data to be extracted from the product information database 116. The store management control uses the relationship data to direct the tree structure control 304 to construct a local data structure representing the hierarchy of groups of an electronic store, thus enabling the tree structure control 304 to render (i.e., draw graphical and textual elements of) the left pane 308 of the store design user interface 310. The store management control 306 also uses extracted relationship information to construct a local data structure associating information about individual products with groups containing them.

Detailed Description Text (28):
A store designer initiates the store design application by using the enhanced Web browser 112 to establish a communication link to the Internet. The enhanced Web browser 112 accesses a Web server 106 hosting an electronic store by transmitting, in part, a URL (Uniform Resource Locator) value (e.g., "http://mystore.design.com") to the Internet which uniquely identifies the Web server hosting the electronic store design application. The Web server 106 responds by transmitting initial HTTP data 302 to the enhanced Web browser 112.

Detailed Description Text (37):
Events recognized by a dialogue box window include a keystroke event and a mouse click event. The keystroke event occurs whenever a key from the keyboard is typed. A mouse click event occurs whenever the mouse pointer is located over the dialogue box and the user presses one of the buttons of the mouse. Event handlers are typically registered for both the keystroke event and the mouse click event.

Detailed Description Text (50):
It will be appreciated by those of ordinary skill in the art that the expand and contract, drag and drop, and double click events recognized by the tree structure control 304 and the store management control 306 may occur as a result of a series of key strokes typed on a keyboard rather than by use of a mouse pointer. For example, an element of either the left pane 308 or the right pane 309 may be selected by pressing the tab key repeatedly until a desired item is highlighted (usually by rendering it in a different color) indicating that it is selected. A double click event may occur by pressing the enter key when an element has been selected using the tab key. Also, drag and drop events may occur by using the tab key to select an element using keyboard keystrokes to activate a pulldown menu, to

select a cut option, and to select a paste option. It will be understood that the present invention is not limited by a user interface method such as a mouse, keyboard, or voice control input.

Detailed Description Text (56):
The Initial_Event_Handler generates a query in the form of a name/value pair. A name/value pair is generated by combining three data elements: (1) the name of a data value, (2) an `=` character, and (3) data representing a value. Some examples of name_value pairs are "population=15,300,250," "temperature=28C," "ForeColor=Blue" and "Baseball_Team=Yankees." One of ordinary skill in the art will appreciate that it is common to communicate data values over the Internet in the form of name/value pairs. The following is one example of a name/value pair representing a query generated by the Initial_Event_Handler: Query=Select Group_Name, Group_, Parent From Relationships, Groups Where ID_Type=`G` And ID=1000 And Relationship=`Contains` And Related_ID_Type=`G` And Related_ID=Group_ID

Detailed Description Text (57):
In the above example, the name of the data value is "Query", followed by an "=", and the remainder constitutes data representing a value. One of ordinary skill will understand that the above query can be generated by a simple reference to a character string constant, or by concatenating one or more character string constants and one or more character string variables. The query associates a value of "G" with the parameter "ID_Type", the value "1000" with the parameter "ID", and the value "Contains" with the parameter "Relationship". If the root level group My Store 320 is represented in the group table 206 by a data record of type "G" (group) having a unique ID value of "1000", it will thus be appreciated that the above query may be used to retrieve data from all data records related to the root level group My Store 320 by a "Contains" (i.e., contained within) relationship.

Detailed Description Text (59):
After generating a query in name/value pair format, at least one executing thread of the Initial_Event_Handler issues a post request 314 (i.e., a request to post an Internet message) in a further step 508 by synchronously calling a Send_Message routine of the enhanced Web browser 112. It is well known in the art that Web browsers include routines similar to the present Send_Message routine, which accept as parameters data in name/value pair format as well as data representing a message type (e.g., a Post message), embed the parameter data in a message, and transmit the message to the Internet in a form compliant with the message type. A post message is one of a number of message types included in the Hypertext Transport Protocol (HTTP) used by Web servers and Web browsers.

Detailed Description Text (60):
In the step 508, the Initial_Event_Handler passes the query in name/value pair format to the Send_Message routine and also passes a parameter indicating that the type of message to send is `Post.` Post messages in accordance with the HTTP protocol are well known in the art and will not be further discussed. Because the Send_Message routine of the enhanced Web browser 112 is called synchronously, no further instructions of the calling thread of the Initial_Event_Handler execute until the Send_Message routine returns a result. Synchronous procedure calls (or function calls) are well understood in the art.

Detailed Description Text (62):
The Parse_HTTP routine 350 launches (i.e., causes a computer to begin executing the instructions of) the ISAPI query application 354 if it is not already running. Also, the Parse_HTTP routine 350 extracts the query from the HTTP Post message in name/value pair format and passes the extracted query to the ISAPI query application 354. One of ordinary skill in the art will understand that Web servers possess parsing routines to extract data parameters from HTTP Post messages in name/value pair format and that applications, such as the ISAPI query application,

may be identified by a portion of a URL.

Detailed Description Text (64):
It will be appreciated that an ISAPI query application, dedicated to perform database operations on product data of a single electronic store, includes a constant (i.e., a predetermined value included in a computer program that does not change when the program is executing) representing the name and location of a product information database.

Detailed Description Text (65):
In another embodiment of the present invention, the ISAPI query application is associated with multiple electronic stores and thus conducts transactions with multiple product information databases. In the case where a single ISAPI query application 354 is associated with multiple stores, a user of the enhanced Web browser 112 selects an electronic store from a list of electronic stores. In this embodiment, the user of the enhanced Web browser 112 is first presented with a display generated from an initial HTML page that lists the available electronic stores. When the user clicks on the name of one of the electronic stores, the Web browser 112 sends a request to the Web server for the store management HTML page for that particular store. The name of the electronic store is sent along with the request as part of the URL. For example, when the store name "BiltRite Hardware" appears in the list of electronic stores, then, if the user clicks on that name, a URL having a store identifier parameter is generated. One example of such a URL is: http::/www.server.com/StoreManager.dll?Store="BiltRite Hardware" Thus, the URL contains information identifying the ISAPI application to call (StoreManager.dll), as well as the parameter Name/Value pair to pass to the ISAPI application (Store="BiltRite Hardware"). The ISAPI query application 354 (e.g., StoreManager.dll) uses the parameter to determine which electronic store to administer.

Detailed Description Text (66):
The Parse_HTTP routine 350 passes the store identifier parameter to the ISAPI query application 354. The ISAPI query application 354 uses the store identifier parameter to determine which product information database 116 to access.

Detailed Description Text (67):
One of ordinary skill in the art will appreciate that a two-dimensional array associates store name values with database identifiers (e.g., DB_Array[1,1] ="BiltRite Hardware," DB_Array[1,2]="BR_Hdwr.db," DB Array[2,1]="Underdog Used CD's," and DB_Array[2,2]="Under_Dog.db"), and will further appreciate that the ISAPI query application 354 performs a look up (e.g., sequential name comparison) operation to obtain a database identifier associated with the store identifier value "BiltRite Hardware." It will also be appreciated that a simple database table associating database identifiers with store identifier values could be used in place of a two-dimensional array to identify a database when given a store identifier value. Furthermore, an operating system registry, such as, for example, the Windows NT registry could also be used to associate database names with store identifiers. The present invention is not limited by any method for associating a store name with a database.

Detailed Description Text (68):
When the appropriate database is identified, a Translate_Query routine 356 of the ISAPI query application 354 translates the query parameter from name/value pair format into a format suitable for submission to the relational database server 114. Those skilled in the art will appreciate that different relational database products require differing query commands. Thus, the translation required may be sophisticated or may be simple depending on the relational database server used. The present invention is not limited by a relational database product or a particular query language.

Detailed Description Text (69):
In a preferred embodiment, the ISAPI query application 354 establishes an ODBC link to a computer hosting the product information database and communicates an SQL-compliant query 323 to a relational database server 114 running on the computer. Those of ordinary skill in the art will appreciate that an ODBC link identifies a specific computer operating in a LAN (Local Area Network) to receive a transmitted SQL command. Thus, a relational database server 114 running on the computer identified by the ODBC link receives the SQL command, queries the product information database 116, and generates a result set 324.

Detailed Description Text (71):
The result set 325 generally comprises data from one or more rows of the group table 206 or the products table 204 which satisfy a query. Each row of a result set 324 typically includes three columns: (1) a name value, (2) an ID value, and (3) a parent status value. After the relational database server 114 generates the result set 325, it passes the result set to the ISAPI query application 354. A Format_Result_Set routine 358 of the ISAPI query application 354 translates the result set 325 into name/value pair format as described above. The following is an example of a result set in name/value pair format: Group_Name=DEPARTMENTS, Group_ID=2000, Parent=Y, Group_Name=STAGING AREA, Group_ID=3000, Parent=Y, Group_Name=ALL PRODUCTS, Group_ID=4000, Parent=Y, Group_Name=ALL GROUPS, Group_ID=5000, Parent=Y, Group_Name=SEARCH RESULTS, Group_ID=6000, Parent=N, Group_Name=SALE PRODUCTS, Group_ID=7000, Parent=N

Detailed Description Text (72):
One of ordinary skill understands that the data values included in the above result set may be extracted from the database tables shown in Table 6 and Table 7 by a query requesting all group data records having a relationship of `C` (i.e., `contained in`) with the group data record having an ID value of 000.

Detailed Description Text (73):
In the example result set above, each Group_Name value (e.g., DEPARTMENTS) comprises descriptive text which may later be formatted and displayed as a label associated with group icons 330, 332, 334, 336, 338, 340 displayed on the store design user interface 310. Each Group_ID value (e.g., 2000) uniquely identifies a group data record in the group table 206 of the product information database 116. The Parent parameter of each group in the result set indicates the existence of any groups or products subordinate to (contained in) the respective group (e.g., whether a group is a parent and thus has subordinate or child groups or products). Thus, a group having a Parent parameter of `Y` contains at least one subordinate group or product, and a group having a Parent parameter of `N` contains no subordinate groups or products.

Detailed Description Text (74):
If there is some error condition created by the query, then the result set data includes an error code representing the specific error, rather than a collection of rows from the products table 204 or the groups table 206. It will be appreciated that numerous error conditions could prevent the success of a database query ranging from a syntax error in the query to a disk volume being off-line and that a unique code is associated with each such error condition. The following are examples of an error condition in name/value pair format: Error=32 or Error=Disk Volume Not Ready. When an error condition prevents completion of a query, a result set comprises a representation of the error condition in name/value pair format.

Detailed Description Text (76):
The ISAPI query application 354 passes the result set data in name/value pair format to a Generate_HTTP_Response routine 352 of the Web server 106. The Generate_HTTP_Response routine 352 generates a response message by combining the result set data in name/value pair format with other data which identifies the enhanced Web browser 112 as the target of an Internet message and which ensures

compliance with the HTTP protocol. The Web server 106 transmits the response message to the World Wide Web 104.

Detailed Description Text (77):
The enhanced Web browser 112 receives the response message, extracts the result set data, and returns the result set data in name/value pair format to the Initial_Event_Handler which, in a step 510, receives the result set data as a return value of the Send-Message routine. The Send_Message routine then terminates. A thread of the Initial_Event_Handler expects to receive the result set data as a response to post 316 (e.g., a response to its earlier post request 314).

Detailed Description Text (80):
The Add_Branch routine of the tree structure control 304 adds a new node to the Group Tree Structure. In the step 514, the Initial_Event_Handler passes four parameters to the Add_Branch routine: (1) the ID value of the root level group, (2) a Group_Name value, (3) a Group_ID value, and (4) a Parent value (e.g., 1000, \`DEPARTMENTS\`, 2000, and \`Y\`). The ID value of the root level group identifies a node that already exists in the Group Tree Structure, and the Add_Branch routine adds a new node subordinate to the identified existing node.

Detailed Description Text (86):
While the query generated in the earlier step 506 is designed to extract group data, the query generated in the step 518 is designed to extract product data representing each product contained in the root level group. In the step 518, as in the step 506, the Initial_Event_Handler formats the query in name/value pair format. The following is one example of a name/value pair representing a query generated by the Initial_Event_Handler to extract product data related to the root level group: Query=Select Product_Name, Product_ID From Relationships, Groups Where ID_Type=\`G\` And ID=1000 And Relationship=\`Contains\` And Related_ID_Type=\`P\` And Related_ID=Product_ID

Detailed Description Text (87):
The query to extract product data is communicated to the product information database in the same manner as the query to extract group data. Thus, in a next step 520, the Initial_Event_Handler makes a post request 314 by calling the Send Message routine and passing to it, as a parameter, the generated query in name/value pair format, as well as a parameter indicating that the message type is \`Post.\` The Send_Message routine then formats the message in HTTP format, including a URL in the message which identifies both the Web server 106 and an ISAPI query application 354. Next, the Send_Message routine transmits the message via the World Wide Web 104 to the Web server 106. The Parse_HTTP routine 350 of the Web server 106 recognizes the reference to the ISAPI query application 354 embedded in the message, launches the ISAPI query application 354, and passes to the ISAPI query application 354 the query in name/value pair format.

Detailed Description Text (88):
The Translate_Query routine 356 of the ISAPI query application 354 translates the query from name/value pair format into a format useful to the relational database server 114. The Translate_Query routine 356 transmits the translated query to the relational database server 114. The relational database server 114 receives the query and queries the product information database 116. One of ordinary skill in the art will appreciate that the query generated by the Initial_Event_Handler in the step 518 causes the products table 204 and the relationship table 202 to be joined.

Detailed Description Text (90):
The relational database server 114 generates a result set from the query, and the Format_Result_Set routine 358 translates the result set into name/value pair format. The following is an example of a result set 325 generated by the relational database server 114 and translated into name/value pair format by the

Format_Result_Set routine 358: Product_Name=Pit Crew T-Shirt, Product_ID=0543, Product_Name=Propeller Head T-Shirt, Product_ID=0544

Detailed Description Text (91):
The result set generated by the query is communicated to the enhanced Web browser 112. Thus, the ISAPI query application 354 passes the result set to the Generate_HTTP_Response routine 352. The Generate_HTTP_Response routine 352 generates a response message by combining the result set data in name/value pair format with additional data which ensures compliance with the HTTP protocol and identifies the enhanced Web browser 112 as a destination. The Web server 106 transmits the response message to the enhanced Web browser 122 via the World Wide Web 104.

Detailed Description Text (92):
The enhanced Web browser 112 receives the response message and passes the result set data in name/value pair format to the Initial_Event_Handler as a result of the Send_Message routine. The Send_Message routine terminates and, in a further step 522, the Initial_Event_Handler receives the result set data. A thread of the Initial_Event_Handler receives the result set data as a response to its post message 316 transmitted to the Web server 106 in the step 520.

Detailed Description Text (94):
Routines of the store management control 306 create, manage, and maintain a binary tree type data structure called, for example, a Product Tree Structure. The Product Tree Structure comprises nodes (e.g., collections of data) each of which includes left and right node pointers to other nodes. Each node also includes data about a group (i.e., a Group_Name value and a Group_ID value), and a pointer to a product data structure. Each product data structure includes information (i.e., a Product_Name value and a Product_ID value) about a product advertised or offered for sale by an electronic store. Each product data structure also includes a pointer which may point to another product data structure.

Detailed Description Text (95):
FIG. 6 illustrates a portion of an example Product Tree Structure. A node 602 includes information about a group having a Group_Name value of `Sedans` and a Group_ID value of 60011. The node 602 also has a left node pointer 604 to a node 606 and a right node pointer 608 to a node 610. The node 602 also has a pointer 612 to a product data structure 614. The product data structure 614 has a Product_Name value of `Honda Accord` and a Product_ID value of 0121, as well as a pointer 616 to a product data structure 618. The product data structure 618 has a Product_Name value of `Toyota Camry` and a Product_ID value of 0122. The product data structure 618 also has a pointer 620 to a product data structure. The pointer 620 is null.

Detailed Description Text (96):
The node 606 has a Group_Name value of `Sports Cars` and a Group_ID value of 60007. Also, the node 606 includes a left node pointer 622 and a right node pointer 624 which are null. A pointer 626 of the node 606 points to a product data structure 628 which has a Product_Name value of `Mazda Miata` and a Product_ID value of 0091. The product data structure 628 has a pointer 630 to a product data structure. The pointer 630 is null.

Detailed Description Text (97):
The node 610 has a Group_Name value of `More Sedans` and a Group_ID value of 60033. Further, the node 610 includes a left node pointer 632 and a right node pointer 634 which are null. A pointer 636 of the node 610 points to a product data structure 638 which has a Product_Name value of `Mercury Sable` and a Product ID value of 0154. The product data structure 638 also includes a pointer 640 which points to a product data structure 642. The product data structure 642 has a Product_Name value of `Olds Aurora` and a Product_ID of 0155. Also, the product data structure 642 has a pointer 644 that is null.

Detailed Description Text (99):
A navigation routine employing recursion receives a parameter identifying a node of a binary tree, examines data values of the node, determines whether data values of the node satisfy search criteria, and, if not, issues a call to itself, passing as a parameter a node pointed to by the left node pointer of the current node. On return from that call, the navigation routine makes another call to itself, passing as a parameter a node pointed to by the right node pointer of the current node.

Detailed Description Text (100):
Such a binary tree navigation routine may be applied to the nodes of the binary tree data structure illustrated in FIG. 6. To search the binary tree for a node whose Group_ID value is 60033, a navigation routine is called and receives as a parameter the node 602. In a first iteration of execution, the navigation routine checks the Group_ID value of the node 602, determines that value to be 60011, and concludes that 60011 is not equivalent to 60033. The navigation routine then calls itself, passing as a parameter the node 606 (whose address is accessible from the left node pointer 604 of the node 602). A second iteration of the navigation routine thus checks the Group_ID value of the node 606, determines the value to be 60007, and concludes that 60007 is not equivalent to 60033. Before the second iteration of the navigation routine attempts to call itself, it checks the left node pointer 622, determines the left node pointer 622 is null (e.g., no nodes exist to the left of the current node) and, thus, does not call itself.

Detailed Description Text (101):
Next, the second iteration of the navigation routine attempts to call itself again using as a parameter a node pointed to by the right node pointer of node 606. Because the right node pointer of node 606 is null, the second iteration of the navigation routine again avoids calling itself, and the second iteration of the navigation routine terminates, returning control to the first iteration of the navigation routine.

Detailed Description Text (102):
The first iteration of the navigation routine then continues executing and calls itself, passing as a parameter a node 610 pointed to by the right node pointer 608. A third iteration of the navigation routine begins running and checks the Group_ID value of the node 610, determines that the value is 60033, and concludes that 60033 is equivalent to the search value 60033. Thus, a recursive navigation routine successfully traverses the nodes of the binary tree illustrated in FIG. 6 to locate a node having a particular ID value.

Detailed Description Text (103):
One of ordinary skill in the art will understand that searching a binary tree structure for a node having a particular numeric value is improved if the binary tree is constructed such that for any given node, its left node pointer points to a node having a numeric value less than its own, and its right node pointer points to a node having a numeric value greater than its own. The present invention is in no way limited by any method of constructing or searching a binary tree structure.

Detailed Description Text (105):
The Initial_Event_Handler, in the step 526, calls the Add_Product routine and passes to it three parameters: (1) the Group_ID value of the root level group, (2) a Product_Name value, and (3) a Product_ID value. The Add_Product routine navigates the Product Tree Structure to search for a node whose Group_ID value matches that of the Group_ID parameter. If no matching node is located, the Add_Product routine creates such a node and links it to the Product Tree Structure. The Add_Product routine then establishes a current node (i.e., the matched node if the search succeeded, the created node if the search failed). The Add_Product routine then determines whether the current node points to any product data structures. If so, the Add_Product routine navigates to the end of the linked list of product data

structures (i.e., finds the first product data structure whose pointer is null). Next, the Add_Product, routine allocates memory for a new product data structure and sets its Product_Name value equal to the Product_Name parameter and its Product_ID value equal to the Product_ID parameter. Finally, the Add_Product routine links the new product data structure to the end of the linked list of product data structures, or, if the current node does not point to any product data structures, the Add_Product routine links the new product data structure to the current node.

Detailed Description Text (107):
The refreshed left pane 308 displays icons and text labels associated with groups subordinate to the root level group represented by the My Store icon 320. As illustrated in FIG. 3, the groups subordinate to the root level group My Store 320 are a departments group 330, a staging area group 332, an all products group 334, an all groups group 336, a search results group 338, and a sale products group 340. In addition, the refresh method of the tree structure control 304 places an expand icon adjacent to group icons representing groups whose Parent parameter is `Y` (i.e., groups which have further subordinate groups or products below them). Such a refresh method for tree structure controls is known in the art. See Microsoft Visual Basic Professional Features 455, Microsoft Corporation, 1995. Expand icons are also known in the art.

Detailed Description Text (109):
In a preferred embodiment, additional special groups are subordinate to a root level group: The staging area group 332 comprises any number of groups or products whose relationships or properties are being modified by a merchant or store designer. The all products group 334 comprises an enumeration of all products stored in the products table 204. Thus, the all products group 334 provides a merchant or store designer convenient access to a list of all products, regardless of their relationship to any group or to each other. Likewise, the all groups group 336 enumerates all groups included in the group table 206, and thus provides convenient access to all groups of an electronic store. Those of ordinary skill appreciate that database commands, such as SQL commands, are easily constructed to retrieve all records of a database table, and further that such a command would facilitate retrieving all data records from the products table 204 to generate a result set. It will be understood that data fields of the result set facilitate creation of data structures underlying an all products group 334 using techniques disclosed herein. The search results group 338 comprises a collection of groups or products resulting from a search request issued by a user. The search results group 338 represents products and groups identified by the most recently performed search. Thus, groups and products identified in each new search replace existing groups and products within the search results group. Those of ordinary skill appreciate that database commands, such as SQL commands, are easily constructed to search the records of a database table, to retrieve the records matching particular query parameters, and to generate a corresponding result set. It will further be understood that many types of interfaces are available to elicit from a user a text or numeric pattern which may contain wildcard (or variable) pattern matching specifications. The Merchant Workbench invokes such an interface when a user clicks on the search command 776 (see FIG. 7D). The user responds by entering search criteria from which the Merchant Workbench generates an SQL query. Finally, it will be understood that a result set generated by a database search command employing pattern matching on particular fields of data records facilitates construction of data structures underlying a search results group 338 using techniques disclosed herein. The sale products group 340 comprises all products currently marked for inclusion in a promotion or sale. Thus, a store designer or a merchant can conveniently access each and every product marked for sale in an electronic store. One of ordinary skill will appreciate that a status field of each data record in the products table 204 may be set to a particular value indicating that the product represented by the data record has been designated for inclusion sale by a merchant. It will be understood that database query commands, such as SQL commands,

can easily be formulated to search for and retrieve data records whose status fields match a predetermined status value and that a corresponding result set would be generated. It will be further understood that such a result set facilitate creation of data structures underlying a sale products group 340 using techniques disclosed herein.

Detailed Description Text (111):
To display information about subordinate products, the refresh method is passed the value of the Group_ID associated with the current group. At the time the initialize event of the store management control 306 is recognized, the root level group (e.g., My Store) is the current group, and the refresh method will search for any products contained in (i.e., directly subordinate to) the root level group. Whether any products are contained in the root level group is a decision made by the designer of the electronic store, and that decision may be dictated by whether the store designer wants any products advertised on an initial Web page of the Web site.

Detailed Description Text (112):
The refresh method of the store management control 306 navigates (i.e., searches) the Product Tree Structure for a node whose Group_ID value matches that of the current group. When the refresh method of the store management control 306 is called by the Initial_Event Handler, the root level is the current selected group, and if its Group_ID value is 1000, then the refresh method navigates the Product Tree Structure to find a node whose Group_ID value is 1000. Once the matching node is found, the refresh method examines the node's pointer to a product data structure. If the pointer is null, then there is no product data to display, and the refresh method terminates.

Detailed Description Text (113):
If, however, the matched node points to a product data structure (e.g., representing a first popular t-shirt), then the Product_ID value (e.g., 0543) and the Product_Name value (e.g., `Pit Crew T-Shirt`) are accessed and displayed as a first product entry 360 in the right pane 309. The refresh method uses standard text display routines known in the art (e.g., such as those commonly used by a refresh method to display entries of a standard list box control) to display Product_ID and Product_Name values. See Microsoft Visual Basic Language Reference 772, Microsoft Corporation, 1995.

Detailed Description Text (114):
The refresh method then examines the pointer of that product data structure which may point to another product data structure. If that pointer is null, then the refresh method terminates. If the pointer points to another product data structure (e.g., representing a second popular t-shirt), then the Product_ID value (e.g., 0544) and the Product_Name value (e.g., `Propeller Head T-Shirt`) of that product data structure are displayed as a second product entry 362. The refresh method continues thus to navigate a linked list of product data structures until a null pointer is encountered and the refresh method terminates.

Detailed Description Text (115):
After displaying information about subordinate products, the refresh method of the store management control 306 displays information about subordinate groups. The refresh method of the store management control 306 invokes a Get_Subordinate_Groups routine of the tree structure control 304 and passes as a parameter the Group_ID associated with the root level group. Tree structure control routines which retrieve subordinate elements of a selected element are known in the art and will not be further discussed. See Microsoft Visual Basic Professional Features 463, Microsoft Corporation, 1995.

Detailed Description Text (116):
The Get_Subordinate_Groups routine returns, in one embodiment of the present

invention, a pointer to a linked list of group structures. Each group structure in the linked list contains information about a group, including Group_ID value and Group_Name value, and also includes a pointer to another group structure. The refresh method of the store management control receives the pointer to this linked list of structures and sequentially navigates the group structures. Navigation of the linked list terminates when a pointer of a group structure is null.

Detailed Description Text (124):
An expand icon 716, represented by a plus sign inside a small box, is located to the left of the departments icon 704. A similar expand icon 718 is located to the left of the staging area icon 706. Expand icons and other command options presented by a user interface are generally activated by using a pointing device, such as a mouse, in combination with depressing a key or button such as a mouse button.

Detailed Description Text (126):
FIGS. 8A and 8B illustrate steps involved in updating the store design user interface 310 upon recognition of an expand event. In a first step 802, an expand event is recognized. In a next step 804, an internal Expand_Event_Handler of the tree structure control 304 begins running. The Expand_Event_Handler of the tree structure control 304 determines, in a further step 806, the Group_ID of the group to be expanded. Expand_Event_Handlers of tree structure controls which return values identifying an element of an hierarchical structure to be expanded are known in the art.

Detailed Description Text (127):
In a further step 808, the Expand_Event_Handler invokes an ExpandGroup_Event_Handler of the store management control 306. The Expand_Event_Handler passes the Group_ID value of the group to be expanded as a parameter to the ExpandGroup_Event_Handler.

Detailed Description Text (128):
In a next step 810, the ExpandGroup_Event_Handler generates a query designed to retrieve data representing all groups that are contained in the group to be expanded. In a preferred embodiment, the query is constructed by concatenating a character string constant, a character string variable including digit characters representing the Group_ID value, and another character string constant. The following is an example of a query generated by the ExpandGroup Event_Handler: Select Group_Name, Group_ID, Parent From Relationship, Groups Where Relationship.ID_Type=`G` and Relationship.ID=Parent_Group_ID and Relationship.Relationship=`Contains` and Relationship.Related_ID_Type=`G` and Relationship.Related_ID_Groups.Group_ID

Detailed Description Text (129):
In a next step 812, the ExpandGroup_Event_Handler calls the Send_Message routine to transmit the query via the Internet. As described in relation to FIG. 5, the Send_Message routine receives the query as a parameter, as well as a message-type of `Post` as a second parameter. Communication of the query to the product information database 116 occurs as was described in relation to FIG. 5.

Detailed Description Text (130):
The relational database server 114 receives the query and queries the product information database 116. The relational database server generates a result set including data retrieved from data records having a `contained in`relationship with the data record. corresponding to the group to be expanded. The ISAPI query application 354 of the Web server 106 formats the result set in name/value pair format. The Web server 106 communicates the result set in name/value pair format to the enhanced Web browser 112 in the manner described in relation to FIG. 5.

Detailed Description Text (132):
The ExpandGroup_Event_Handler passes four parameters to the Add_Branch routine: (1)

the Group_ID value corresponding to the group to be expanded, (2) a Group_Name value, (3) a Group_ID value; and (4) a Parent value. The Add_Branch routine then creates a new node and adds that node to the Group Tree Structure in the manner described in relation to FIG. 5.

Detailed Description Text (134):
In a further step 824, the ExpandGroup_Event_Handler calls the Send_Message routine to transmit the query via the Internet. As previously described in connection with FIG. 5, a result set is generated from the query and formatted in name/value pair format and communicated from the Web server 106 to the enhanced Web browser 112.

Detailed Description Text (135):
In a next step 826, the ExpandGroup_Event_Handler receives the result set comprising data retrieved from product data records and formatted in name/value pair format. In a next step 828, the ExpandGroup_Event_Handler determines whether the result set is empty. If not, then, in a further step 830, the ExpandGroup_Event_Handler calls the Add_Product routine to add data representing each product in the result set to the Product Tree Structure in the manner described in connection with FIG. 5.

Detailed Description Text (137):
In a next step 836, the ExpandGroup_Event_Handler calls the refresh method of the store management control 306 which navigates the Product Tree Structure to locate a node corresponding to the group that was selected to be expanded. When that node is located in the Product Tree Structure, the refresh method navigates the linked list of product data structures and outputs one row (e.g., text strings representing a Product_ID value and a Product_Name value) to the right pane display 309 for each product data structure encountered until a null pointer is reached.

Detailed Description Text (146):
The right pane 742 displays the products and groups contained in the sedans group in a two-column format. Each column is designated by a column heading--a Product_ID heading 744, and a Name heading 746. A first product 750 contained in the sedans group is represented in FIG. 7D as having a Product_ID "P0121," and a name "Honda Accord." A second product 752 contained in the sedans group is represented as having a Product_ID "P0122," and a name "Toyota Camry." A more sedans group is represented as being contained within the sedans group by an icon 754 labeled "More Sedans." Also, a luxury sport utility group is represented as being contained within the sedans group by an icon 756 labeled "Luxury Sport Utility."

Detailed Description Text (150):
Each element (e.g., text string or icon), whether associated with a group or a product, is enabled as a valid drag source. Thus, each element representing a product (e.g., text string representing a Product_ID value or Product_Name value) can be selected by a mouse and `dragged` to a different position in the store design user interface 310. Furthermore, each such element is associated with information about the respective product it represents and that information is available to event_handler routines when such element is dragged.

Detailed Description Text (151):
Each element of the left pane 902 or the right pane 904 associated with a group (e.g., an icon or a text string representing a Group_Name) can also be selected with a mouse and dragged to a different location in the store design user interface. When a group element is dragged, information about the group (e.g., Group_ID value and Group_Name value) is accessible by event handlers.

Detailed Description Text (153):
FIG. 10a illustrates steps performed when a drag event occurs in the left pane 902. In a first step 1002, a left pane drag event is recognized. In a next step 1004, an event handler called, for example, L_Drag_Event_Handler, of the Tree Structure

Control 304, begins running. In a further step 1006, the L_Drag_Event_Handler determines the Group_ID value of the group associated with the icon being dragged. Also, in the step 1006, the L_Drag_Event_Handler determines the Group_Name value and the Parent value of the group associated with the icon being dragged. In a next step 1008, the L_Drag_Event_Handler terminates.

Detailed Description Text (155):
FIG. 10b illustrates steps performed when a drag event is recognized in the right pane 904. In a first step 1010, a drag event in the right pane is recognized. An event handler called, for example, R_Drag_Event_Handler, of the store management control 306 begins running in a further step 1012, following the recognition of the drag event in the right pane 902. In a next step 1014, the R_Drag_Event_Handler determines whether the element in the right pane being dragged represents a group or a product. To determine whether a dragged element represents a group or product, the R_Drag_Event_Handler accesses the drag source information made available by the drag source object. If a dragged element represents a group, then the R_Drag_Event_Handler accesses drag source information including Group_ID value, Group_Name value, Parent value, and a Type value. A Type value of `G` indicates, for example, that the dragged element represents a group. If a dragged element represents a product, however, then the R_Drag_Event_Handler accesses drag source information including Product_ID value, Product_Name value, and a Type value of `P` (indicating that the dragged element represents a product).

Detailed Description Text (156):
In a next step 1016, the R_Drag_Event_Handler determines the ID value of the product or group being dragged. Also, in the step 1016, the R_Drag_Event_Handler determines the group name or the product name of the element being dragged, as well as the Parent value if the element is a group. In a further step 1018, the R_Drag_Event_Handler tenninates.

Detailed Description Text (159):
FIGS. 11A and 11B illustrate steps performed to modify relationships between groups or products when a drop event is recognized in either the left pane 902 or the right pane 904. In a first step 1102, a drop event is recognized in the left pane upon the release of a mouse button that was depressed to begin a drag operation. In a next step 1104, an event handler called, for example, Drop_Event_Handler, of the store management control 306 begins running. In a further step 1106 the Drop_Event_Handler determines the Group_ID value of the group associated with the dropped target icon.

Detailed Description Text (160):
Then, in a next step 1108, the Drop_Event_Handler generates a database command to add a new data record to the relationship table 202. The database command adds a record having an ID field equal to the Group_ID associated with the drop target icon, and a Related_ID equal to the Group_ID value or the Product_ID value corresponding to the drag source element (the element dragged by the user). The new data record will also specify a `contained in` relationship because the command to add a new data record specifies that the value of the Relationship field be set to `C`.

Detailed Description Text (162):
In a further step 1112, the Drop_Event_Handler determines whether a data record was successfully added to the relationship table 202 of the product information database 116 by examining a value of a result code embedded in a message transmitted by the Web server 106 to the enhanced Web browser 112. The Drop_Event_Handler receives a result code of, for example, "Result=Success" if a data record was successfully added; a result code of, for example, "Result=Non-Fatal Error" if a non-fatal error prevented the addition of a new data record; or a result code of, for example, "Result=Fatal Error" if a fatal error prevented the addition of a new data record.

Detailed Description Text (164):
Then, in a next step 1118, the Drop_Event_Handler generates a database command to remove a data record from the relationship table 202. The database command is designed to remove a data record having a Related_ID field equal to the Group_ID value associated with the dragged icon. The SQL command to remove the data record from the relationship table 202 requires certain parameters: the ID of the object to move (passed as a parameter called Moved_Object_ID), the Group_ID of the parent group (group where the group is moved FROM, passed as a parameter called From_Group_ID). The following is an example of such a database command: Delete Relationship Where Relationship.ID_Type=`G` And Relationship.ID=From_Group_ID And Relationship.Relationship=`Contains` And Relationship.Related_ID=Moved_Object_ID

Detailed Description Text (165):
In one embodiment of the present invention, no Product_ID value is the same as any Group_ID value. One skilled in the art will understand that, in this embodiment, the database command required to remove a data record from the relationship table 202 need not specify whether the Related_ID field of the data record to remove comprises a Product_ID value or a Group_ID value.

Detailed Description Text (167):
In a further step 1122, the Web server 106 transmits a result message to the enhanced Web browser indicating whether the database command successfully removed a data record from the product information database 116. The result message includes a result code in name/value pair format. When a data record is successfully removed, the result code is, for example, "Result=Success." When a data record is not removed due to a non-fatal error, the result code is, for example, "Result=Non-Fatal Error." When a fatal error prevents a data record from being removed, the result code is, for example, "Result=Fatal Error." In the step 1122, the Drop_Event_Handler examines the result code, and, if the result code indicates that a data record was not successfully removed, then in a next step 1124, the Drop_Event_Handler determines whether the error was fatal or non-fatal. If the error was fatal or if three consecutive non-fatal errors occurred, then, in an additional step 1126, the Drop_Event_Handler terminates. If the error was non-fatal and no more than two consecutive non-fatal errors occurred, then, the Drop_Event_Handler records an additional non-fatal error and repeats step 1120.

Detailed Description Text (168):
If, in the step 1122, the Drop_Event_Handler determines that a data record was removed successfully, then in a next step 1128, the Drop_Event_Handler accesses the drag source information to determine whether the user dragged an element representing a group. If so, then, in a next step 1130, the Drop_Event_Handler calls a Remove_Branch routine of the tree structure control 304.

Detailed Description Text (169):
The Drop_Event_Handler passes the Group_ID associated with the dragged element (e.g., an icon representing a group) as a parameter to the Remove_Branch routine. The Remove_Branch routine removes a node from the Group Tree Structure whose Group_ID value matches that associated with the dragged icon.

Detailed Description Text (170):
In a next step 1132, the Drop_Event_Handler calls the Add_Branch routine of the tree structure control 304. The Add_Branch routine receives as parameters (1) the Group_ID value associated with the drop target icon, (2) the Group_Name associated with the drag source element, (3) the Group_ID value associated with the drag source element, and (4) the Parent Value associated with the drag source element. The Add_Branch routine then adds a new node to the Group Tree Structure, the new node having a Group_Name, Group_ID, and Parent Value equal to that of the group represented by the drag source element.

Detailed Description Text (172):
If, in the step 1128, the Drop_Event_Handler determines that the dragged element represents a product, then, in a next step 1140, the Drop_Event_Handler calls the Remove_Product routine of the store management control 306. The Drop_Event_Handler passes to the Remove_Product routine the Group_ID value associated with drop target icon as a first parameter and the Product_ID value associated with the drag source element as a second parameter. The Remove_Product routine navigates the Product Tree Structure, locates a node whose Group_ID value matches that associated with the drop target icon, and establishes that node as a current node. Then, the Remove_Product routine accesses a pointer of the current node which points to a linked list of product data structures. The Remove_Product routine navigates the linked list of product data structures until it encounters a product data structure having a Product_ID that matches the Product_ID associated with the drag source element. The Remove_Product routine then removes that product data structure from the linked list of product data structures and terminates. One of ordinary skill in the art understands how to remove a data structure from a linked list of such data structures.

Detailed Description Text (173):
In a next step 1142, the Drop_Event_Handler calls the Add_Product routine of the store management control 306 and passes three parameters: (1) the Group_ID value associated with the drop target icon, (2) a Product_Name value associated with the dragged element, and (3) a Product_ID value associated with the dragged element. The Add_Product routine navigates the nodes of the Product Tree Structure and locates the node whose Group_ID matches that associated with the drop target icon. If no such node exists, the Add_Product routine adds such a node to the Product Tree Structure. The Add_Product routine then establishes the located (or created) node as a current node. Next, the Add_Product routine accesses a pointer of the current node which points to a linked list of product data structures, navigates that linked list to its end, allocates memory for a new product data structure, and adds the new product data structure to the linked list. The Add_Product routine sets the Product_ID value and Product_Name value of the new product data structure to the values as passed in the second and third parameters, which correspond to values associated with the dragged element. The Add_Product routine then terminates.

Detailed Description Text (174):
In a next step 1144, the Drop_Event_Handler updates the Parent value of a node in the Group Tree Structure whose Group_ID matches that of the drop target icon. Updating this Parent value ensures that the Group Tree Structure represents that a group associated with the drop target icon has at least one subordinate group or product. Thus, when the left pane display is updated, an expand icon is associated with the drop target icon, indicating that groups or products subordinate to the drop target icon exist and may be examined.

Detailed Description Text (183):
The new group dialogue box 1301 is displayed in response to selection of the new group command 772. A merchant or store designer enters information into the fields of the new group dialogue box, except for the merchant ID field 1302 and the Group_ID field 1304, for which the Merchant Workbench generates field values automatically. The value of the merchant ID field 1302 is held constant for all groups offered by one merchant. The value generated for the Group_ID field 1304, however, is a unique value (i.e., no two groups of products offered by a merchant have the same Group_ID value).

Detailed Description Text (184):
A user (such as a merchant operating an electronic store) enters a group name value in the Group_Name field 1306, and enters the name of an HTML template file in the template file field 1308. The HTML template file is thereby associated with the group being created. A merchant describes the new group by entering description

text in the description field 1310. Also, a merchant can associate a graphical image with the new group by entering the name of a graphic file in the small image field 1312.

Detailed Description Text (186):
A user selects the okay button 1316 to finalize entry of values in the group name field 1306, template file field 1308, description field 1310, and large image field 1312. When the okay button 1316 is selected, computer instructions associated with the okay button 1316 perform steps to create a new data record in the group table 206.

Detailed Description Text (187):
FIG. 14 illustrates steps performed to add a new data record to the group table 206. In a step 1402, the data values entered in the new group dialogue box 1301 are organized and formatted as parameter values to be included in a database command. In a next step 1404, a database command is generated which is designed to create a new data record in the group table 206. Information required to add a new record to the group table 206 includes a current merchant ID, a new group ID, a new group name, a group description, a template file name, and a small image name. The following is one example of an SQL command to add a record to the group table 206: Insert into Groups Values(CurrentMerchantID, NewGroupID, NewGroupName, `N`, Description, ", Today'sDate, Today'sDate, ", ", ", ", TemplateFileName, SmallImageName)

Detailed Description Text (189):
In a next step 1408, a result message is received from the Web server 106 indicating the success or failure of the database command. The result message is passed as a parameter to a Message_Box routine which, in the step 1408, displays the result message in a read only dialogue box on the user's computer screen. Such Message_Box routines are known in the art. In a further step 1410, instructions associated with the okay button 1316 examine the result message to determine whether a data record was successfully added to the group table 206. If not, then, in a further step 1412, the instructions terminate. If so, then, in a next step 1414, the instructions of the okay command 1316 determine whether the user selected a parent group.

Detailed Description Text (191):
If, in the step 1414, it is determined that a parent group was not selected, the instructions of the okay button 1316 terminate in the step 1412. If, however, a parent group was selected, then, in a next step 1416, a database command is generated that is designed to add a new record to the relationship table 202. The database command specifies, for example, a Related_ID field equal to the ID value generated for the new group, a Relationship field equal to `C` (e.g., `contained in`), and an ID field equal to the Group_ID of the parent group.

Detailed Description Text (194):
The Merchant Workbench automatically generates a value for the merchant ID field 1322 which is constant for every product offered by a particular merchant. A merchant enters a Product_ID value 1324 and a Product_Name value 1326. The merchant also enters a unit price 1328 and a unit size value 1330. The merchant determines the effective time period for the product by entering a value in the effective date field 1332 and determine when a product will expire by entering a date in the expiration date field 1334.

Detailed Description Text (195):
A merchant associates an HTML template file with the new product by entering the name of such a template file in the template file field 1336. A merchant describes a new product by entering a description in the description field 1338 and by entering a shorter description in the short description field 1340. A merchant enters detail information about a new product by entering a value in the detail

field 1342. Also, by entering the name of a graphic file in the large image field 1344, a merchant associates a picture of a product with the other information about the new product. Also, a merchant associates a small picture or thumbnail-size picture of a product with the new product by typing the name of a graphic file comprising a small illustration in the small image field 1346.

Detailed Description Text (198):
Another type of relationship can be created to support cross sales. A cross sale occurs when a consumer buys a product of one type and also decides to buy a different, but related product (e.g., a consumer buying a pair of shoes also buys socks, or a consumer buying toothpaste also buys a toothbrush, or a consumer buys french fries along with a hamburger). To facilitate such cross sales, a merchant entering information about a new product can select a cross-sales option, resulting in the presentation of a list of existing products. The merchant can then select one or more related products from a list of existing products. Once one or more related products have been selected and the merchant clicks the okay button 1350, a new record is added to the relationship table for each product selected. Each such record has a Related_ID field equal to the Product_ID of the new product, a Relationship field value of `CS` (i.e., cross sale) and an ID field equal to the related product selected. Techniques for using such cross sale relationships include automatically generating a message to a consumer who has just ordered a particular product, such message displaying a list of related products and asking the consumer if the consumer would like to order one of the listed products.

Detailed Description Text (199):
One of ordinary skill will appreciate that validation is performed on the fields entered into either a new group dialogue box 1301 or a new product dialogue box 1320. Such validation includes determining, for example, whether values entered to represent HTML template files correspond with existing files, whether entered graphic files exist, and whether date or price values are properly formatted. Furthermore, one of ordinary skill understands that Merchant_ID values, Group_ID values, and Product_ID values may be generated automatically or entered by a user.

Detailed Description Text (202):
FIG. 15 illustrates steps performed to update information about a group or a product. In a first step 1502, a double-click event is recognized in the right pane 742. In a next step 1504, a DblClk_Event_Handler begins running in response to the double-click event. In a further step 1506, the DblClk_Event_Handler accesses the Group_ID value or the Product ID value associated with the element that was double-clicked by the user. It is known in the art to access a data structure associated with an icon when a user clicks the icon using a mouse pointer. One of ordinary skill will appreciate that such a data structure contains a value, such as a Group ID or Product ID, by which the data structure is distinguished from other such structures.

Detailed Description Text (203):
In a step 1508, the DblClk_Event_Handler generates a database command designed to retrieve all fields of the data record having an ID value equal to the Group_ID value or Product_ID value associated with the double-clicked element. An example of a command to retrieve all fields of a group record is "Select * from Groups where Groups.Group_ID=SelectedGroupID", and an example of a command to retrieve all fields of a product record is "Select * from Products where Products.Product_ID=SelectedProductID". In a next step 1510, the DblClk_Event_Handler calls the Send_Message routine to transmit the database command to the product information database 116.

Detailed Description Text (204):
In a further step 1512, the DblClk_Event_Handler receives a result set in name/value pair format in the manner described in relation to FIG. 5. The result set includes values for all the fields of a product data record or a group data

record, depending on whether the user clicked an element representing a product or an element representing a group.

Detailed Description Text (205):
In the step 1512, the field values of the result set are used to fill in fields of a dialogue box. Again, the dialogue box includes all the fields of a product data record if a user double-clicked a product element, or the dialogue box contains all the fields of a group data record if the user double-clicked a group element. In a next step 1514, the DblClk_Event_Handler displays the dialogue box on the graphical user interface of the enhanced web browser 112.

Detailed Description Text (206):
In a next step 1516, a user edits the fields of the displayed dialogue box. In a further step 1518, the DblClk_Event_Handler determines whether the user selects the `okay` button or the `cancel` button. If the user selects the cancel button, then in a next step 1520 the DblClk_Event_Handler terminates. If, however, in the step 1518, the DblClk_Event_Handler determines that the user selects the okay button, then, in a further step 1522, the DblClk_Event_Handler generates a database command to store the updated field values of the dialogue box as modified by the user. Then, in a further step 1524, the database command to store the updated field values is transmitted to the product information database 116. One of ordinary skill in the art will appreciate that error checking is performed to verify the successful result of the step 1510 and the step 1524.

Detailed Description Text (207):
In a step 1526, the DblClk_Event_Handler terminates. It will be thus understood that the elements of the right pane 742 act as hyperlinks which, when double-clicked, enable store designers to examine and modify fielded data values associated with any product or group.

Detailed Description Text (210):
In an additional step 1606, a user organizes the hierarchy of groups and products. To organize the hierarchy of groups and products, a user manipulates elements, such as icons or text strings, of the left pane 740 or right pane 742 of the store design user interface 760, as described in relation to FIG. 9.

Detailed Description Text (220):
In an additional step 1612, a store designer enters rules for determining which pages and which products are preferred by a particular consumer. In the step 1612, the store designer selects the preferred rules command 778 of the store design user interface 760. A Preferred_Rules routine accesses a preferred rules file. The preferred rules file includes two records: the first record comprises a collection of paired values representing preferred page rules, the second record comprises a collection of paired values representing preferred product rules. One of ordinary skill appreciates that there are many methods for storing paired values in a record of a file and the present invention is not limited by any such method.

Detailed Description Text (221):
Each paired value of the preferred page rules includes a first value representing a minimum number of accesses to a particular page by a particular consumer and a second value representing a period of time. A preferred page rule is satisfied when a consumer accesses a particular page at least as many times as specified by the first value within the time period specified by the second value. It will be appreciated by one of ordinary skill that a consumer's preference for a Web page may be shown by criteria other than a number of accesses over a period of time. For example, a consumer's preference for a Web page may be shown by the length of time (e.g., in minutes or hours) the consumer has spent accessing a Web page. This length of time may be monitored both during a single shopping session at an electronic store as well as across all shopping sessions by the consumer. Each paired value of the preferred product rules includes a first value representing a

minimum number of products ordered and a second value representing a period of time. A preferred product rule is satisfied when a consumer orders a number of units of a product as least as large as the first value within the time period specified by the second value. It will be understood that a consumer's orders for products may show a preference not only for products, but also for groups (or categories) of products. Thus, a preferred group rule may include, for example, a first value representing a minimum number of products ordered from a group and a second value representing a period of time.

Detailed Description Text (222):
The Preferred_Rules routine generates a preferred rules dialogue box and displays the dialogue box on the store design user interface 760. The preferred rules dialogue box displays a preferred page list box comprising a list of entries. Each entry includes two values of a preferred page rule. The preferred rules dialogue box also displays a preferred product list box comprising a list of entries. Each entry includes two values of a preferred product rule. The entries in the preferred page and preferred product list boxes are extracted from the preferred rules file. It is known in the art to access values in a file and generate a list box of entries where each entry corresponds to a value from the file.

Detailed Description Text (223):
The store designer, in the step 1612, modifies entries of or adds entries to the preferred page and preferred product list boxes as desired. When the store designer selects an `okay` button of the preferred rules dialogue box, the Preferred_Rules routine saves the entries of the preferred page list box as paired values in the first record of the preferred rules file and saves the entries of the preferred product list box as paired values in the second record of the preferred rules file.

Detailed Description Text (224):
One of ordinary skill will understand that many alternative methods exist for updating values stored in a file, and therefore that the present invention is not limited by any method of modifying the paired values of preferred rules.

Detailed Description Text (231):
In a preferred embodiment, the HTML page engine 126 passes the query in a generic form to a relational database server 114, which translates the query into a specific form and queries the product information database 116.

Detailed Description Text (237):
The query is then posed against the relationship table 202 of the product information database 116. As illustrated in FIG. 18, three rows 1818, 1820, 1822 of the relationship table 202 have a "contains" relationship with a group having the ID 60004. Each of the rows 1818, 1820, 1822 satisfying the query respectively includes a Rel_ID value 1824, 1826, 1828 (e.g., related ID value) identifying a group related to the automotive group.

Detailed Description Text (238):
To generate a result set from the query, the Rel_ID values 1824, 1826, 1828 are used to locate data records in the group table 206. A name and a template file name are extracted from each located data record in the group table 206. Thus, as illustrated in FIG. 18, a name of "Sedans" and a template file name of "Sedans.html" are extracted from a data record 1830 having an ID value 60011. Likewise, the names "Sports Car" and "Sport Utility" and template file names "Sportsc.html" and "Sportu.html" are extracted from data records 1832, 1834 having ID values 60012 and 60013 respectively. An example result set is the following: Sedans, sedans.html Sports Car, sportsc.html Sport Utility, Sportu.html

Detailed Description Text (245):
The two other data records 1914, 1916 have Rel_ID values P0121 and P0122 both

identifying data records of the product table 204. Accordingly, two data records 1920, 1922 are retrieved from the product table 204 having respective ID values P0121 and P0122. The name "Honda Accord" and the template file name "sedancarha.html" are extracted from one data record 1920 of the product table 204, and the name "Toyota Camry" and the template file name "sedancartc.html" are extracted from another data record 1922. The following is an example result set: More Sedans, msedans.html Honda Accord, sedancarha.html Toyota Camry, sedancartc.html

Detailed Description Text (247):
The HTML page engine creates an HTML text file 1924 by removing the query script 1910 from the text file 1908 and replacing the query script 1910 with the HTML coded result set 1926. Accordingly, one of ordinary skill understands that selectable hypertext links of a Web page are extracted from the product information database 116. It will be understood that various embodiments of the present invention extract differing elements associated with groups or products, such as graphic files comprising illustrations of a product or text files comprising detailed or summary descriptions of groups or products or numeric values representing available units or prices. Parameters of query scripts embedded in template files determine what information is extracted from a product information database. These extracted elements are translated into HTML coded result sets and merged with a template file to create an HTML compliant file which is transmitted for presentation to a consumer.

Detailed Description Text (248):
In one implementation of the Merchant Workbench, before transmitting the HTML text file 1924 to the Web browser, the Web server queries the product information database to examine the availability status of each product. One of ordinary skill will understand that a product ID value may be used to query an availability status field associated with each product. If it is determined that a product is not available, then the hyperlink associated with the unavailable product is removed from the HTML text file. It will thus be appreciated that a product availability query permits a single product information database to support both electronic store product sales and physical store sales.

Detailed Description Text (257):
To recognize individual consumers and distinguish between them, the software tool generates a unique ID value for each consumer. Thus, a unique consumer ID value is assigned to each individual consumer.

Detailed Description Text (258):
To effect such assignments, the Web server 106 constructs a persistent client state cookie (`cookie`) and sends the cookie to a consumer's Web browser 102. The cookie comprises a name/value pair, such as `Consumer_ID=00333714.` After a Web browser 102 receives such a cookie, the Web browser 102 transmits the particular name/value pair (e.g., Consumer_ID=00333714) to the Web server 106 with every Web page request. Thus, when any page of an electronic store is accessed, the Web server 106 identifies the requesting consumer. Persistent client state cookies are known in the art. The present invention is not limited, however, by any method for identifying a consumer. For example, in another embodiment of the invention, a Web page of an electronic store prompts a consumer to supply a name, password, or other identification information upon each access to the electronic store. The present invention uses the supplied identification information to identify each consumer accessing the electronic store.

Detailed Description Text (259):
FIGS. 20A and 20B illustrate steps performed to assign a consumer ID to a consumer and to log a consumer's access to a Web page or a consumer's order for a product. In a first step 2002, the Web server 106 receives a request from a Web browser 102 for a Web page and scans the request message for a cookie identifier. In a

preferred embodiment, a cookie identifier for a consumer is a name/value pair beginning with the name, "Consumer_ID=" followed by a value assigned to the consumer.

Detailed Description Text (260):
In a next step 2004, the Web server 106 determines whether a cookie identifier exists in the request message. If not, then a further step 2006, the Web server 106 generates a consumer ID value to uniquely identifying the consumer. It is well known in the art to successively generate values uniquely identifying each of a set of elements. One such method is to store an initial value on non-volatile storage media, read the value and increment it to generate a unique value, and then replace the stored value on the non-volatile storage media with the incremented value.

Detailed Description Text (261):
Next, in an additional step 2008, the Web server 106 generates a set-cookie command. A set-cookie command comprises a keyword, "Set-Cookie:" followed by a number of possible parameters. The set-cookie command uses a first parameter comprising a name/value pair. To generate a name/value pair, the Web server 106 combines an identifier constant (e.g., "Consumer_ID=") with the unique value (e.g., "000333714") generated in the step 2006. In the step 2008, the Web server combines the "Set-Cookie:" keyword with the name/value pair to generate the set-cookie command (e.g., "Set-Cookie: Consumer_ID=00333714").

Detailed Description Text (263):
In a next step 2012, the Web server determines whether the consumer requested the first Web page (or "welcome" page) of an electronic store. This step 2012 follows the prior step 2004 if the Web server locates a cookie identifier in the request message. It will be understood that consumers browsing an electronic store routinely access the welcome page at the beginning of each shopping session. Accessing a welcome page is a shopping behavior common to virtually all consumers, and such access reveals no particular preference on the part of any consumer. Thus, one embodiment of the present invention does not compile information detailing each consumers' access to a welcome page. To determine whether a consumer is accessing a welcome page, the Web server 106 scans the URL of the request message for the presence of a file name that matches the file name of the welcome page. It will be appreciated that a file name for a welcome page may be "mystore.htm" and that a request message having a URL value of "http:.backslash.www.elecstore.com.backslash.mystore.htm" comprises a request for a welcome page of an electronic store.

Detailed Description Text (265):
Generally, to order a product from a Web-based electronic store, a consumer enters purchase information into an order form Web page which includes text entry fields prompting a consumer, for example, for number of units, payment method (e.g., credit card number), and shipping address. A Web server 106 determines that a consumer has ordered a product by (1) scanning the URL of a request message for the presence of a file name corresponding to an order form Web page, and (2) validating the purchase information (e.g., credit card number, whether inventory includes at least the number of units ordered, etc.) entered by the consumer which is included in the request message. It will be understood that a file name of an order form Web page may be "order.asp" (in one embodiment of the present invention, the file extension ".asp" designates a template file having an embedded script that can be processed) and that a URL value of "http:.backslash..backslash.www.elecstore.com.backslash.order.asp" indicates that a consumer has accessed an order form Web page. It will be further understood that, if insufficient inventory is available, a Web page may be transmitted to the consumer describing that the consumer's order could not be processed for insufficient inventory, or displaying some other explanatory message.

Detailed Description Text (266):

If, in the step 2014, the Web server 106 determines that the consumer ordered a product, then, in an additional step 2016, the Web server creates a new data record for the product order table of the traffic analysis database. The Web server scans the consumer's cookie identifier to determine the Consumer_ID uniquely identifying the consumer. The Web server 106 also accesses the purchase information entered by the consumer on the order form Web page which is included in the request message. The Web server 106 scans the request message for a product name, or product identifier value, and also for a quantity value (i.e., the number of units orders).

Detailed Description Text (267):
The Web server, in the step 2016, generates a database command designed to add a record to the product order table. One of ordinary skill understands that such a database command includes values for the fields of the new record. The Web server supplies values for each of the following fields: Consumer_ID established by scanning the consumer's cookie identifier Product_ID established by scanning the request message generated by a Web browser to order a product; the Product_ID is always specified in the request message URL Quantity established by scanning the request message generated by a Web browser to order a product Date established by accessing a common calendar function of the computer operating system under which the Web server 106 operates Time established by accessing a time function of the computer operating system under which the Web server 106 operates

Detailed Description Text (268):
Then, in the step 2016, the Web server 106 issues the database command to the traffic analysis database to create a new record in the product order table. In one embodiment, the Web server uses the Quantity value to generate a database transaction command which, when processed, decreases in the product information database the total number of units available with respect to the product ordered. A status field, units field, or other attribute field of a record in the products table 204 may represent the number of units available for a particular product. Also, in the step 2016, the Web server 106 queries the relationships table 202 to retrieve any records having a "CS" or cross sales relationship with the products ordered by the consumer. The Web server 106 uses values from the Related_ID fields of any retrieved records to query the products table 204 for description information for any cross sale related products. The Web server then creates an association between the information describing any cross sale related products and the consumer. Next, in the step 2020, the Web server 106 scans the URL of the request message for the name of a template file.

Detailed Description Text (269):
If, in the step 2014, the Web server determines that the consumer did not order a product, then, in a next step 2018, the Web server 106 generates a database command designed to add a new record to the browse table of the traffic analysis database. It will be understood that such a database command accepts parameters representing values for the fields of a new record of the browse table. To supply a value for the Consumer_ID field of the new record, the Web server 106 access the consumer's cookie identifier and extracts the unique Consumer_ID value. The Web server establishes a value for the Template_File field of the new record by extracting a template file name from the URL of the request message. One of ordinary skill will appreciate that a URL of "http:.backslash..backslash.www.elecstore.com.backslash.auto.htm" includes a template file "auto.htm" and that methods for extracting such a template file name are known. The Date and Time fields are established by accessing, respectively, a common calendar routine and a common clock routine of the operating system under which the Web server 106 operates.

Detailed Description Text (273):
If, however, in the step 2026, the HTML page engine determines that a customize reference exists in the template file, then, in a further step 2030, the HTML page

engine queries the browse table of the traffic analysis database for all records having a Consumer_ID field matching the Consumer_ID value of the consumer's cookie. The resulting set of data records describes all prior accesses by the consumer to any of the Web pages of the electronic store.

Detailed Description Text (274):
In a next step 2032, the HTML page engine examines the resulting set of data records to determine whether the consumer accessed any Web page with sufficient frequency to create additional hyperlinks to the page for the benefit of the consumer. Threshold access frequencies are stored as a set of preferred page rules. One of ordinary skill will understand that Web page access frequencies may be expressed as a certain minimum number of accesses within a certain time period. Thus, a Web page access frequency may comprise two values: (1) an access total (i.e., an integer representing the number of times a consumer accessed a particular Web page), and (2) a time period (e.g., an integer representing a number of days). Accordingly, a set of preferred page rules comprises one or more paired values, and each particular preferred page rule comprises one pair of values.

Detailed Description Text (280):
ID of each preferred page. In a next step 2036, the HTML page engine uses the Template_ID value of each preferred page to query the group table 206 or the product table 204 for a Group_Name value or a Product_Name value, respectively, associated with the Template_ID.

Detailed Description Text (281):
Then, in an additional step 2038, the HTML page engine combines the Template_ID value and either a Group_Name value or a Product_Name value to create HTML hyperlink tags referencing preferred pages. For example, if a Template_ID is 'sedans.html` and an associated Group_Name is `Sedans`, the HTML page engine creates the HTML hyperlink tag: <A HREF="/web/sedans.html">Sedans</A>

Detailed Description Text (283):
After the HTML page engine creates HTML hyperlink tags, or if, in the step 2032, the HTML page engine located no preferred pages, then, in a next step 2040, the HTML page engine queries the product order database for all records having a Consumer_ID value that matches the Consumer_ID of the consumer's cookie. After extracting all data records from the product order table describing product orders by the consumer, then, in a further step 2042, the HTML page engine determines whether any preferred product rules are satisfied.

Detailed Description Text (284):
It will be understood that a preferred product rule, like a preferred page rule described above, comprises a pair of values: the first value representing a number of times a particular product was ordered by the consumer, and the second value representing a period of time. Thus, a preferred product rule is expressed as a pair of numbers. For example, the preferred product rule (3, 10) is satisfied when a consumer orders at least three units of a particular product within a ten-day period. It will be understood that preferred product rules may be entered, under one method, by selecting the preferred rules command 778 of the store design user interface 760, whereupon a dialogue box is displayed on the store design user interface 760. The dialogue box allows a store designer to delete or alter existing preferred product rules or add new ones. The present invention is not limited by any method of entering preferred product rules.

Detailed Description Text (285):
The HTML page engine, in the step 2042, scans all the data records describing product orders by the consumer and creates a list of Product_ID values, each Product_ID representing a product the consumer has ordered. Beginning with the first Product_ID in the list, the HTML page engine determines whether the product was ordered with sufficient frequency to satisfy a preferred product rule, thus

making the product a preferred product of the consumer. One of ordinary skill will appreciate that the HTML page engine examines the Quantity field value of each data record when determining how many units of a particular product were ordered in a particular time period.

Detailed Description Text (286):
In an alternative embodiment, preferred group rules are used. In this embodiment the HTML page engine combines the quantities of products within each group to generate a value for each product group indicating the number of products from each group that the consumer has ordered. It will be understood that each such value generated may represent the number of products of each group ordered on each day by the consumer. The HTML page engine determines whether products of any group were ordered with sufficient frequency to satisfy a preferred group rule.

Detailed Description Text (287):
If, in the step 2042, the HTML page engine determines that at least one product is a preferred product of the consumer, then, in a further step 2044, the HTML page engine extracts a Product_ID value from a product order table data record for each preferred product. The HTML page engine then uses each Product_ID value to query the product table 204 for an associated Template_ID value and an associated Product_Name value.

Detailed Description Text (288):
In a next step 2046, the HTML page engine combines the Template_ID value and Product_Name value associated with each preferred product to create an HTML hyperlink tag for each preferred product. For example, if a Template_ID value is "hacksawb.html" and a Product_Name value is "Hacksaw Blade", then the HTML page engine creates the HTML hyperlink tag: <A HREF="/web/hacksawb.html">Hacksaw Blade</A>Thus, the HTML page engine creates an HTML hyperlink tag for each preferred product. One of ordinary skill in the art will appreciate that, when preferred group rules are used, a similar hyperlink tag may be generated in the same manner to associate a Template_ID value with a Group_Name value.

Detailed Description Text (291):
The Web server 106 scans the request message 2102 for a URL 2104 and a cookie identifier 2106. The Web server 106 then scans the URL 2104 for the name of a template file (e.g., "sedan.html") and also scans the cookie identifier 2106 for a Consumer_ID value (e.g., "55714") uniquely identifying the consumer.

Detailed Description Text (293):
The HTML page engine then uses the Consumer_ID value of the cookie identifier 2106 to query data records from both the browse table and the product order table of the traffic analysis database. A set of data records describing all page accesses by the consumer is extracted from the browse table, and a set of data records describing all product orders placed by the consumer is extracted from the product order table.

Detailed Description Text (296):
Next, the HTML page engine examines the data records extracted from the product order table to determine whether, for this consumer, there are any preferred products. Four data records 2122, 2123, 2124, 2125 of the product order table describe orders placed by the consumer for a particular product (e.g., a hacksaw blade). The HTML page engine accesses preferred product rules 2126 and finds two such rules: (20, 60) and (10, 30). The HTML page engine determines that the first rule is not satisfied because the consumer did not order 20 units of the product within 60 days. However, the second rule is satisfied because, the sum of the value of the quantity fields of the four records 2122, 2123, 2124, 2125 is 10 and all of the four orders were placed within a 30-day period.

Detailed Description Text (297):

After finding a preferred product for the consumer, the HTML page engine constructs an HTML hyperlink tag 2128 for the product. The HTML page engine uses the Product_ID value for the preferred product to query the product table 204 for a Template_ID value (e.g., "hacksawb.html") and a Product_Name value (e.g., "Hacksaw Blade").

Detailed Description Paragraph Table (15):
Consumer_ID (value uniquely identifying a consumer) Template_File (representing a Web page accessed by a consumer) Product_ID (value identifying product ordered) Group_ID (value identifying product category for ordered product) Date (date Web page was accessed) Time (time Web page was accessed)

Detailed Description Paragraph Table (16):
Consumer_ID (value uniquely identifying a consumer) Product_ID (value uniquely identifying a product) Quantity (number of units ordered) Date (date order was placed) Time (time order was placed)

CLAIMS:

6. A system for generating web pages of an electronic store, comprising: a web browser configured to run on a first computer; a web server configured to run on a second computer; a product information database including information describing a relationship between a group of products, said product information database comprising a relationship table that describes a relationship among a group of products, and a group table that is referenced by the relationship table, the group table containing data records that are identifiable by an ID value and further containing a product name and a template file name, said product information database stored on a computer storage media, said web server having access to said product information database; a first web page including a first hyperlink identifying a template file stored on said computer storage media, said template file including content data complying with an Internet protocol and a database query command; an Internet message including information identifying said template file, said Internet message generated by said web browser and sent over the Internet to said web server in response to a user selecting said first hyperlink; and a second web page generated by said web server in response to receipt of said Internet message, said web page including said content data and also including result data generated by querying said product information database using said database query command, said result data containing at least one product name and at least one corresponding template file name.

Previous Doc Next Doc Go to Doc#